Universidad Autónoma de Madrid
Escuela politécnica superior



Universidad Autónoma de Madrid

Grado en Tecnologías y Servicios de Telecomunicación

TRABAJO FIN DE GRADO

# Aprendizaje auto supervisado para reconocimiento de objetos

Autor: Alejandro Camacho Valladares
Tutor: Marcos Escudero Viñolo
Ponente: Jesús Bescós Cano

JULIO 2020

# Aprendizaje auto supervisado para reconocimiento de objetos

Autor: Alejandro Camacho Valladares
Tutor: Marcos Escudero Viñolo
Ponente: Jesús Bescós Cano

**Video Processing and Understanding Lab**
**Dpto. Tecnología Electrónica y de las Comunicaciones**
**Escuela Politécnica Superior**
**Universidad Autónoma de Madrid**
**Julio de 2020**

Trabajo parcialmente financiado por...

# Resumen

Hoy en día, las técnicas basadas en aprendizaje profundo o *Deep Learning* han alcanzado rendimientos inimaginables en múltiples tareas. La cantidad de datos disponibles con los que poder entrenar estos modelos está aumentando rápidamente. La mayoría de los datos actuales no están etiquetados, imposibilitando el análisis supervisado. Generar datos etiquetados es un proceso costoso. Como alternativa, surgen los modelos basados en el aprendizaje auto-supervisado.

En el ámbito de visión artificial, en numerosas ocasiones (p.e. en el ámbito de imagen médica), el dominio en el que se encuentran los datos no es convencional (imágenes digitales en color RGB capturando escenas desde la perspectiva humana) y por tanto debemos adaptar los modelos adecuadamente. El objetivo de este Trabajo de Fin de Grado es estimar la viabilidad de obtener beneficio de las tareas auto-supervisadas y adaptarlas a imágenes en dominios no convencionales, en nuestro caso serán imágenes dermatológicas de lesiones de piel.

Para ello, empezamos realizando un estudio del estado del arte acerca de los conceptos básicos que usan las redes neuronales, en concreto las redes neuronales convolucionales. Posteriormente, describimos los diferentes tipos de aprendizaje que se pueden llevar a cabo en las redes neuronales y específicamente el aprendizaje auto-supervisado (*Self-supervised learning*) y sus técnicas.

Seguidamente, se explican los entornos desarrollados utilizados y los conjuntos de datos con los que se ha trabajado. Además, se explica la arquitectura de la red neuronal convolucional usada (ResNet), así como de las técnicas sobre las que se construye el TFG (*transfer learning* y *fine tuning*) y el desarrollo base de nuestro trabajo.

En el siguiente capítulo, se analizan y evaluan los resultados de las pruebas realizadas con los modelos del capítulo anterior. Para ello, se realiza una comparativa con los resultados obtenidos.

Por último, basándonos en los resultados obtenidos se llevarán a cabo unas conclusiones objetivas y se detallarán trabajos futuros.

## Palabras clave

*Deep Learning*, redes neuronales convolucionales, aprendizaje auto-supervisado, técnicas auto-supervisadas, lesiones de piel, tarea pretexto,*transfer learning*, *fine tuning*.

# Abstract

Nowadays, Deep Learning-based techniques have developed unimaginable performances on multiple tasks. The amount of data available with which to train these models is increasing rapidly. Most of the current data is unlabeled, making supervised analysis impossible. Generating labeled data is a costly process. As an alternative, models based on self-supervised learning emerge.

In the field of artificial vision, on numerous occasions (eg in the field of medical imaging), the domain in which the data is located is not conventional (RGB color digital images capturing scenes from a human perspective) and therefore we must adapt the models appropriately. The objective of this Final Degree Project is to estimate the benefit of self-supervised tasks and adapt them to images in non-standard domains, in our case they will be dermatological images of skin lesions.

To do this, we start by making a state-of-the-art study was conducted on the basic concepts used by neural networks, specifically convolutional neural networks. Afterwards, we describe the different types of learning that can be carried out by neural networks and specifically self-supervised learning and its techniques.

Then, the developed environments and the datasets with which we have worked are explained. In addition, we will explain the architecture of the convolutional neural network used (ResNet), as well as the techniques on which the TFG is built (transfer learning and fine tuning) and the base development of our work.

In the next chapter, we will analyze and evaluate the results of tests performed on the models in the previous chapter. For this, a comparison is made with the results obtained.

Finally, based on the obtained results, objective conclusions will be made, and future work will be detailed.

## Key Words

Deep Learning, convolutional neural networks, self-supervised learning, self-supervised techniques, skin lesions, pretext task, transfer learning, fine tuning.

# Agradecimientos

*En primer lugar, quiero dar las gracias a mi tutor, Marcos Escudero Viñolo, por ayudarme durante todo el desarrollo del Trabajo de Fin de Grado y por la dedicación y apoyo que ha tenido en mi cuando lo he necesitado.*

*También agradecer a mi familia el apoyo que me han dado durante esta etapa, en especial a mis padres que siempre han confiado en mí y a mi abuela que siempre me ha enorgullecido y dado fuerzas.*

*A todas las personas que han aparecido en mi vida en estos años de carrera. En especial a Agrupamiento Primario, el equipo de fútbol de la universidad, a Sara, Belén y Fer que siempre han estado a mi lado en los momentos más difíciles.*

*Finalmente, dar las gracias a mi grupo de amigos externo por todos los momentos de desconexión, consejos y apoyo.*

*Alejandro Camacho Valladares*

*Julio 2020*

# Índice general

# Índice de Figuras

# Índice de Tablas

# 1 Introducción

## 1.1. Motivación

Actualmente muchas de las tareas de clasificación de imágenes están etiquetadas y presentan buenos resultados (aprendizaje supervisado), pero recolectar manualmente estas etiquetas es costoso y difícil de ampliar. Sin embargo, manejar los datos sin etiquetar (aprendizaje no supervisado) no es fácil y los modelos entrenados con estas técnicas funcionan, generalmente, de manera mucho menos eficiente a los etiquetados.

Una de las técnicas de aprendizaje no supervisado es el aprendizaje auto-supervisado. En ella, el objetivo es obtener etiquetas para datos no etiquetados y manejar de manera supervisada conjuntos de datos sin supervisión a través de una tarea de aprendizaje supervisada que se maneja de forma especial para predecir un subconjunto usando el resto.

Gracias al uso de la tarea auto-supervisada (Figura 1.1), conocida como tarea pretexto, se pretende aprender variables latentes de alta calidad como puede ser un clasificador de reconocimiento de objetos con pocas muestras etiquetadas. El aprendizaje de representación auto-supervisada se enfoca en aprender características útiles y efectivas para muchas tareas.

La motivación de este trabajo justamente viene dada por la aplicación de esta técnica de aprendizaje a un conjunto de datos. En concreto, a conjuntos en dominios no convencionales que en nuestro caso son imágenes de lesiones de piel.

En la mayoría de ocasiones, las lesiones de piel son complicadas de detectar, incluso para profesionales involucrados en el ámbito. Por ello, el objetivo es gracias a las técnicas de aprendizaje auto-supervisado clasificar los diferentes tipos de lesiones de piel.

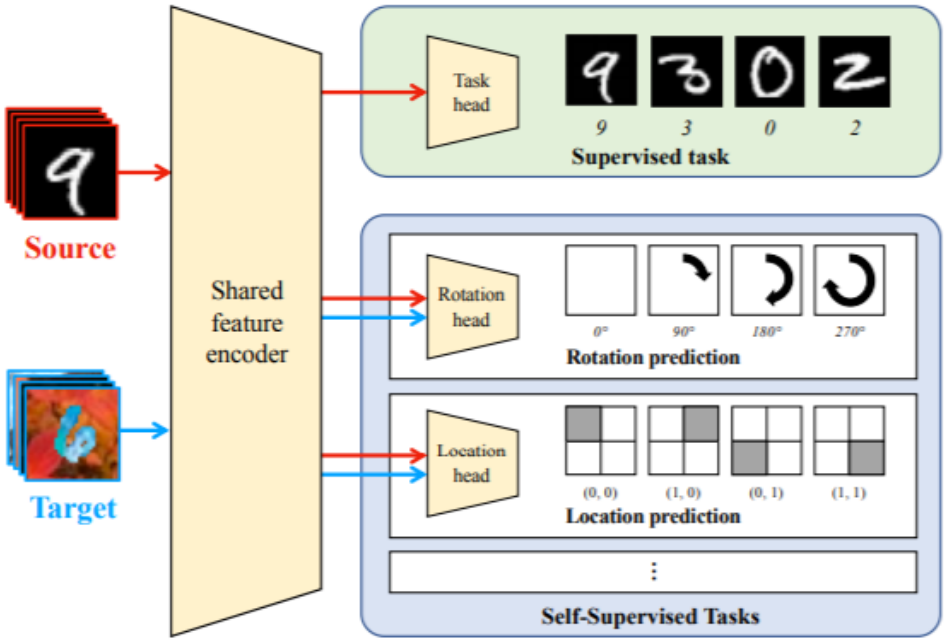



Figura 1.1: Ejemplo que entrena conjuntamente una tarea supervisada con datos etiquetados y otra auto supervisada sin etiquetas. Las tareas usan funciones de alto nivel a través de un codificador compartido, donde aprenden a alinear las distribuciones de características. Extraído de [1].

## 1.2. Objetivos

El reto de este Trabajo de Fin de Grado es calcular la viabilidad de que las técnicas de aprendizaje auto-supervisado son efectivas y que gracias a las tareas pretexto los modelos pueden aprender características generales de las muestras. Además, analizar si, para dominios no convencionales, el hacer uso de esta técnica puede apuntar hacia buenos resultados.

El objetivo final mencionado puede dividirse en varios objetivos parciales que han ido surgiendo mientras se ha desarrollado el trabajo:

- Análisis y estudio del estado del arte.
- Desarrollo del algoritmo base que se utiliza para entrenar el modelo y la definición de la tarea pretexto elegida y las tareas objetivo evaluadas.
- Evaluación de las pruebas.

## 1.3. Organización de la memoria

El presente trabajo se organiza de la siguiente manera:

- Capítulo 1. Introducción del Trabajo de Fin de Grado en el que se definen las motivaciones y los objetivos.
- Capítulo 2. Se explican los conceptos básicos que usan las redes neuronales y donde veremos las posibles arquitecturas que hay. Analizaremos más en detalle las redes neuronales convolucionales las cuales se implementan en la práctica de este trabajo. Por último, se repasan los distintos tipos de aprendizaje que presentan estas redes, donde se destaca y explica en detalle el aprendizaje auto-supervisado y sus técnicas.
- Capítulo 3. Se visualizan los entornos de desarrollo con los que el trabajo ha sido realizado. Por otra parte, se hará un análisis de las bases de datos utilizadas, sus características y como se han manejado y ajustado a nuestro problema a resolver. También analizaremos brevemente la elección de la arquitectura de nuestra red neuronal convolucional y como se ha adaptado a las bases de datos utilizadas. Además, se analizan las técnicas de *transfer learning* y *fine tuning*.
- Capítulo 4. Se realizan y evaluan los experimentos partiendo de las bases de datos y técnicas aplicadas mencionadas anteriormente donde se representan y analizan gráficamente los resultados.
- Capítulo 5. Conclusiones acerca de los resultados obtenidos y posibles trabajos futuros a implementar.
- Bibliografía.

# 2

# Estado del arte

## 2.1. Introducción al Deep Learning

El aprendizaje profundo o *Deep Learning* es generalmente realizado mediante el uso de las redes neuronales o *Neural Networks*, las cuales son utilizadas en diversas estructuras donde su objetivo es un aprendizaje automático a través de una serie de capas con las cuales se aprenden tareas objetivo.

La idea principal del funcionamiento de estas redes está inspirada en la organización neuronal del cerebro humano, en concreto en las neuronas cerebrales que están conectadas unas con otras transmitiendo y recibiendo información. De esta manera, entendemos las redes neuronales como un sistema en el cual tenemos unas señales de entrada, las cuales pasan a través de las capas internas en donde se hacen transformaciones, para que una vez la información llegue a la salida, el resultado sea lo más aproximado posible a uno deseado.

El *Deep Learning*[2] realiza estas tareas a través de una red neuronal artificial compuesta por múltiples capas definidas en el proceso de creación de la red. En primer lugar, siempre tenemos la capa inicial de la red, posteriormente, se envía la información a la siguiente capa. En cada capa la información se vuelve más compleja pero la red es capaz de modelar relaciones más complejas. Estas relaciones vienen determinadas por los pesos de la red, que son el objeto del aprendizaje.

## 2.2. Redes Neuronales

Una red neuronal puede tener el número de neuronas y capas que queramos y puede ser lo compleja que se quiera. Cuanto más compleja sea la red, supuestamente mayor facilidad tendrá en modelar tareas de alto nivel, pero a su vez el entrenamiento será un proceso cada vez más costoso tanto en recursos computacionales como en tiempo y necesidad de datos de entrenamiento. Cuando las redes neuronales presentan muchas capas intermedias, se las engloba dentro del ámbito del *Deep Learning*. (Ver Figura 2.1)

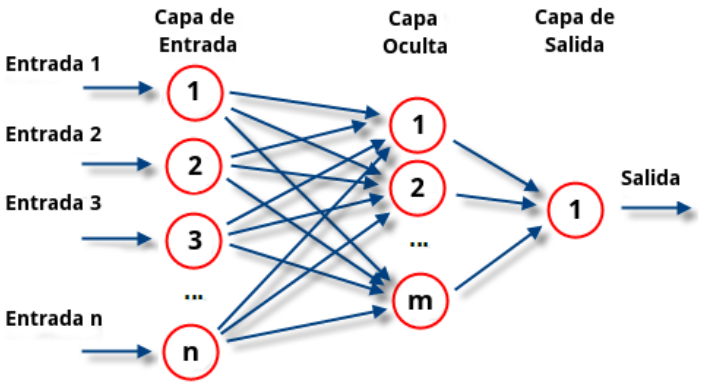



Figura 2.1: Representación de una red neuronal. Extraído de [2].

La información que hay en cada capa son números, y a dicha información se la denomina pesos. El principal problema que existe es que podemos tener una cantidad de parámetros muy grande, por ello lo que se necesita para que la red aprenda es encontrar los pesos óptimos que mejor se ajusten aunque resulte una tarea complicada.

Para controlar si nuestra red aprende o no, utilizamos la función de pérdida de la red la cual se encarga de comparar cómo de cerca estamos del resultado que deseamos. Intentando que la función de error sea mínima ajustando para que el resultado (la salida de la red) cada vez sea lo más parecido al deseado. Para obtener los gradientes de la función de coste a cada peso, podemos recorrer la red en sentido contrario y utilizar los cómputos de gradientes de las últimas capas para obtener los de las capas intermedias, y así hasta la entrada de la red. A este algoritmo se le denomina propagación atrás (*backpropagation*).

## 2.3. Conceptos básicos

### 2.3.1. Conjuntos de datos y fases

Para dar comienzo al entrenamiento de las redes neuronales es necesario dividir el conjunto de datos en dos subconjuntos que se utilizará en las siguientes fases [3]:

- **Fase de entrenamiento** (*train*): Se utiliza el primer subconjunto de datos, datos de entrenamiento. En esta fase se hallan los pesos que definirán la red neuronal. Se calculan de manera iterativa con el objetivo de minimizar el error entre la salida obtenida y la deseada.
- **Fase de prueba** (*test*): Se utiliza el segundo subconjunto de datos, datos de prueba. Se realizan las pruebas con imágenes diferentes a las de entrenamiento con el objetivo de determinar si el modelo calculado anteriormente es adecuado. En algunas ocasiones, puede que el modelo de entrenamiento haya sufrido un sobreajuste (*overfitting*), aprendiendo únicamente características específicas de las imágenes. Por ello, es recomendable utilizar un tercer subconjunto de datos, datos de validación (*validation*), para controlar el aprendizaje de la red neuronal.

### 2.3.2. Funciones principales

En el apartado introductorio se ha visto la estructura básica de las redes neuronales. En este apartado se especifica con más detalle las funciones que llevan a cabo.

### Función de entrada (Función de propagación)

Es la encargada de transformar todas las entradas combinándolas en una sola entrada global a partir de las demás entradas y los pesos.

Normalmente la función usada para unir las entradas es la suma de las entradas por los pesos independientes a cada entrada.

Existen otras reglas de propagación como la distancia euclídea (se usa el cuadrado), distancia de Manhattan (hace uso del valor absoluto) y el máximo entre las entradas.

### Función de activación

Las neuronas (tanto biológicas como artificiales) no solo transmiten la entrada que reciben, hay un paso intermedio donde hay una función de activación que nos dice cuál es el potencial de acción que usa, es decir cuánto de activa está la neurona.

La función de activación [4] utiliza la salida de la función de entrada que como hemos visto es la combinación de las entradas y la transforma hacia la salida. Por una parte, tenemos la combinación de los pesos, $w_i$ con sus respectivas entradas, $x_i$, y por otro lado tenemos b. A este término se le denomina sesgo (bias), el cual mide como de lejos están las predicciones de los resultados.

$$z = b + \sum_i w_i x_i) \tag{2.1}$$

Hay muchas funciones de activación, pero de entre todas ellas destacan 2:

- **Función Sigmoide:** Históricamente, es la más popular y antigua, aunque actualmente han perdido popularidad. Esto es debido a que se complica el entrenamiento en las redes neuronales con muchas capas debido al desvanecimiento de gradiente.

  A las neuronas que utilizan esta función se las denomina neuronas sigmoideas.

  Su función es comprimir, reduciendo la salida a un rango de 0 a 1.

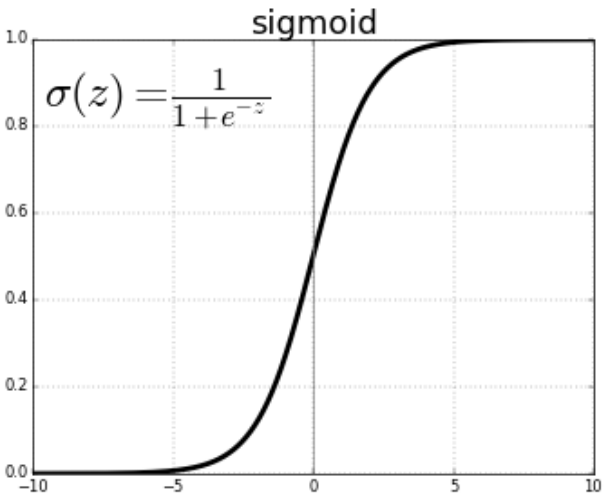



Figura 2.2: Función de activación sigmoide.Extraído de [4].

- **Función ReLU (*rectified linear unit*):** Hoy en día es la más usada a pesar de que existen otras funciones de activación más recientes. Su función es asignar a 0 todos los valores negativos y dejar el mismo valor que tenían para los valores positivos.

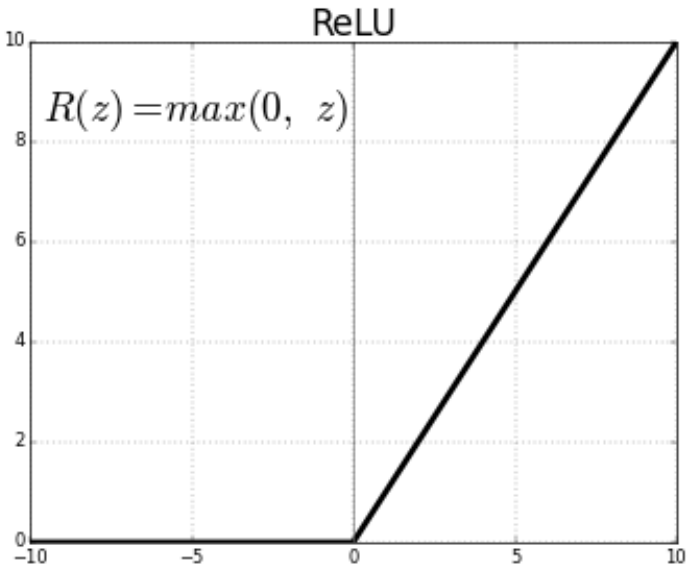



Figura 2.3: Función de activación ReLU.Extraído de [4].

#### Función de salida

Normalmente está función es la propia función de activación y en múltiples ocasiones no es considerada. Se encarga de transmitir a la salida el estado de la neurona actual a la próxima neurona asociada.

### 2.3.3. Descenso por gradiente (*Gradient Descent*)

Una parte esencial de las redes neuronales son los algoritmos de optimización. Estos son los encargados de reducir el error en cada una de las neuronas. Una buena elección hace que la red sea más efectiva tanto en el aprendizaje como en el tiempo.

El descenso por gradiente [5] es uno de los algoritmos de optimización iterativa más reconocidos. El gradiente es una técnica que permite ajustar los parámetros de la red minimizando el error a la salida.

En el uso del descenso por gradiente, se introducen todos los datos disponibles a la vez. Esto puede provocar que las variaciones sean mínimas debido al cálculo del gradiente con todos los datos. Por ello, es conveniente que exista aleatoriedad en la entrada.

Existe una variación del algoritmo mencionado, descenso por gradiente estocástico, SGD (*Stochastic Gradient Descent*), el cual en lugar de utilizar todas las muestras a la vez, introduce aleatoriamente una sola muestra por iteración. Además de introducir aleatoriedad, se consigue que no exista estancamiento en las variaciones. El problema de esta versión del algoritmo es que al tener muchas iteraciones (una por cada muestra), el proceso se vuelve muy lento. Para agilizar el proceso, se introducen las muestras en mini-lotes (*mini-batches*).

En primer lugar, con los datos del conjunto de entrenamiento se introduce un mini-lote con N muestras aleatorias etiquetadas y se realizan los cálculos para obtener las predicciones de las entradas. A este paso se le conoce como *forward propagation*.

La función de coste o pérdida (calcula la diferencia entre la salida de la red y la real) será elegida previamente. La función será evaluada con el mini-lote anteriormente escogido. Se calcula el gradiente de la función de coste para todos los parámetros de la red como la derivada multivariable. (Ver Figura 2.4)

Matemáticamente hablando, el gradiente es la pendiente de la tangente de la función de coste, que es un vector que indica la dirección y sentido donde esta función aumenta más rápido.

Moviéndose en sentido opuesto a este vector sería lo correcto entonces para minimizar esta función.

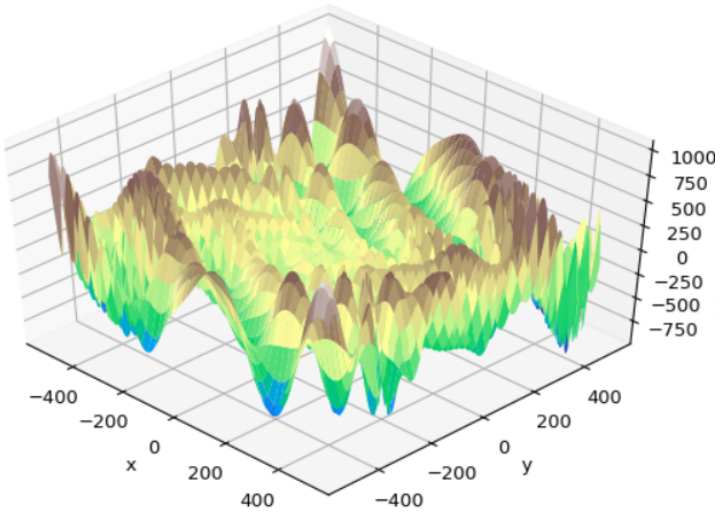



Figura 2.4: Ilustración bidimensional de una función de coste. En general se trabaja en millones de dimensiones (una por parámetro). El cálculo del vector gradiente en una red neuronal profunda no es trivial. Extraído de [5].

Debido a la gran cantidad de parámetros que presentan las redes neuronales es difícil calcular el gradiente. Por ello se dispone del algoritmo de propagación hacia atrás (*backpropagation*) el cual consiste en calcular las derivadas parciales de la función de coste pero con los parámetros de la última capa. Una vez se obtienen las derivadas, se pasa a la capa anterior y se realiza el mismo proceso pero con los parámetros de esta capa. Este proceso se repite hasta llegar a la primera capa de la red neuronal.

Por último, se actualizarán los parámetros de la red, realizando la diferencia entre el valor actual con el obtenido del gradiente por la tasa de aprendizaje ya que se busca ir en sentido contrario al gradiente.

Para evitar que el algoritmo se desestabilice, es importante una buena elección de la tasa de aprendizaje. (Ver Figura 2.5)

El valor de la tasa no debe ser ni muy bajo ni muy alto. Además, es recomendable que vaya variando (disminuyendo) con el tiempo, evitando oscilaciones y mínimos locales.

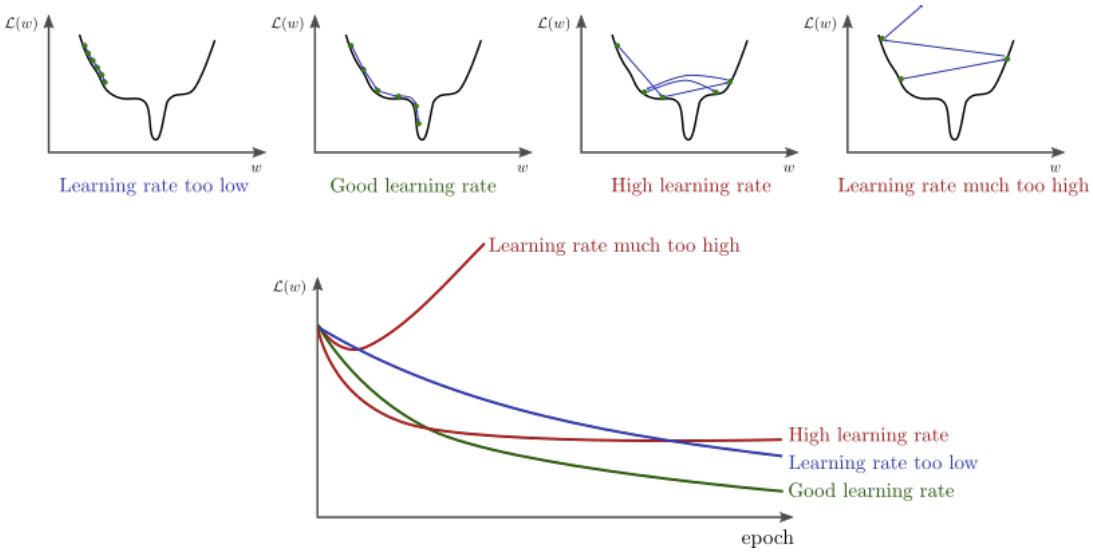



Figura 2.5: Representación gráfica de las curvas de pérdida/épocas según el learning rate impuesto. La curva de color verde es la ideal ya que presenta una buena convergencia. Extraído de [5].

## 2.4. Arquitecturas de las Redes Neuronales

Hay diferentes arquitecturas de redes neuronales artificiales [2] dependiendo del tipo de entrada con el que se trabaje. A continuación, describiremos las más importantes.

- **Redes neuronales prealimentadas:** Este tipo de redes fueron las primeras en ser creadas y utilizadas ya que son el modelo más sencillo. La información solo se propaga en una dirección (hacia delante). No hay ciclos ni bucles y se pasa por tanto de los nodos de entrada a los intermedios y posteriormente a los de salida. Este tipo de redes suele ser utilizado para problemas de clasificación simple.
- **Redes neuronales recurrentes (RNN):** Están inspiradas en el lóbulo temporal del cerebro que es el encargado de la memoria a largo plazo. Este tipo de redes presentan bucles de retroalimentación permitiendo que la información continúe durante todo el proceso, permitiendo hacer uso de información del pasado. Funcionan como réplicas de si misma donde la información se transmite a sus sucesores. Suelen ser utilizadas para problemas como reconocimiento de voz.
- **Redes neuronales convolucionales (CNN):** Están relacionadas con la corteza visual del cerebro. Este tipo de redes se caracteriza en que las entradas son imágenes y destacan por ser efectivas en tareas como son la segmentación y clasificación de imágenes. Como su nombre indica, este tipo de redes poseen capas convolucionales [6]. En la entrada, la red obtiene los píxeles de las imágenes donde normalmente son W x H (ancho x alto) y si tiene más de un canal se le añade D (número de canales). La idea principal es filtrar las imágenes, reducir su tamaño y clasificarlas. Principalmente las redes neuronales convolucionales enfocadas a tareas de clasificación siguen la misma estructura de capas:
  - Capa convolucional: Filtrado de imágenes.
  - Capa de reducción (Pooling): Reducción de parámetros.
  - Capa clasificadora totalmente conectada (Fully Connected): Clasificación del resultado final.

## 2.5. Redes Neuronales Convolucionales (CNN)

### 2.5.1. Capa Convolucional

El proceso [7] consiste en coger una porción de píxeles de la imagen de entrada y realizar un producto escalar con un filtro (*kernel*) elegido que se irá desplazando por todos los píxeles de la imagen de tal forma que obtendremos una matriz con nuevos resultados (Ver Figura 2.6), llamada matriz de activación. Cada píxel de la nueva matriz de salida es la combinación lineal de varios de los píxeles de la imagen de entrada.

Hay 2 parámetros importantes dentro de estas capas a destacar:

- *Kernel*: Es el filtro que se aplica las imágenes y realiza las transformaciones citadas con los píxeles. Su función es obtener patrones y características de las imágenes. Principalmente es de menor tamaño que la imagen. Cuando la imagen es en color, se usarán 3 *kernels* idénticos (RGB, uno para R, *Red*, otro para G, *Green* y el último para B, *Blue*) que se sumarán para obtener la imagen de salida. El desplazamiento del filtro se puede establecer, a este desplazamiento se le denomina *stride*.
- *Padding*: Es una operación la cual añade píxeles con valor 0 en los bordes de la imagen. Estos píxeles de valor nulo se añaden en columnas y en filas. El objetivo principal es obtener una imagen de salida con el mismo tamaño que la imagen de entrada, aunque también se usa para obtener una mayor información de las esquinas (el filtro pasa más por el centro de la imagen).

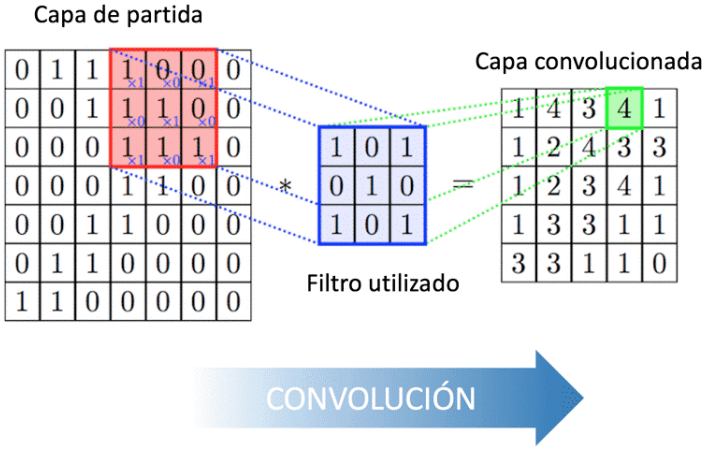



Figura 2.6: Ejemplo del proceso realizado en una capa convolucional. Extraído de [7].

### 2.5.2. Capa de normalización (*Batch Normalization*)

Es un método el cual normaliza los valores de los píxeles de las imágenes. Las imágenes a color suelen tener valor que van de 0 a 255, con este método se pretende normalizar los valores a un rango de 0 a 1[8]. Esta transformación se hace siempre antes de que los datos pasen por la función de activación, así los datos siempre estarán normalizados. Con este método conseguimos reducir el tiempo de entrenamiento de la red ya que se reduce la búsqueda de la línea de decisión.

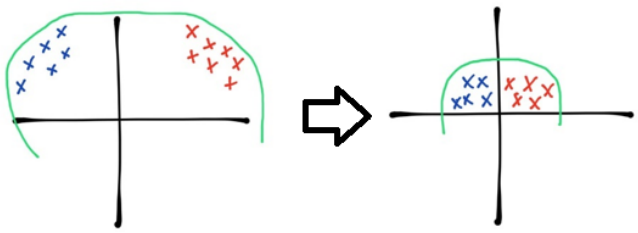


Figura 2.7: Representación de muestras antes y después de normalizar. Extraído de [8].

### 2.5.3. Capa de reducción (*Pooling*)

La utilidad principal de esta capa es reducir la resolución de las imágenes, generalmente no afecta a la dimensión de profundidad [9]. Se pierde información en este proceso, pero no es algo malo ya que en esta capa nos quedamos con la información más importante y con las características y parámetros generales (Ver Figura 2.8). También es beneficiosa esta pérdida de información debido a que se disminuye el tiempo de cálculo en capas posteriores y se reduce el problema del sobreajuste (*overfitting*).

Hay dos métodos que destacan en la reducción de parámetros:

- **Máximo (*Max-pooling*):** Se elige el máximo valor de una ventana de la matriz de activación previamente obtenida.
- **Media (*Average-pooling*):** Mismo método pero el valor a escoger es la media de la ventana.

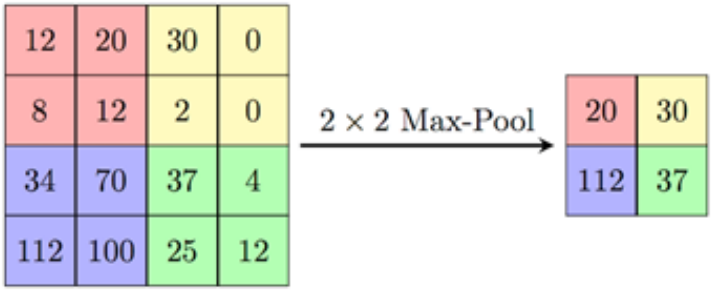



Figura 2.8: Representación explicativa sobre la capa de reducción. En esta imagen se muestra un Max-pooling. Extraído de [9].

### 2.5.4. Capa clasificadora (*Fully Connected*)

Las neuronas de esta capa están totalmente conectadas unas con otras. En esta capa es donde se dará lugar la clasificación de imágenes, donde se clasificarán las imágenes por clases. La salida tendrá el mismo número de neuronas como de clases de quiera predecir.

## 2.6. Tipos de Aprendizaje

El aprendizaje en las redes neuronales se adquiere a través de un entrenamiento de sus parámetros o pesos, donde normalmente el algoritmo consiste en ir calculando los pesos de la red e irlos optimizándolos hasta que la respuesta sea la adecuada.

Sin embargo, hay diferentes tipos de aprendizajes [10] entre las redes neuronales, ya que no todas aprenden de la misma forma y los patrones que siguen son distintos. Destacaremos tres tipos de aprendizaje:

- **Aprendizaje supervisado:** Es un aprendizaje controlado [11]. Consta tanto de los patrones de entrada como de los deseados a la salida, pudiendo así modificar los pesos para adaptar la entrada a la salida. Tenemos el control de la red, pudiendo corregir esta cuando la salida no sea la adecuada. En este tipo de aprendizaje se dice que los datos están etiquetados.

  Este tipo de aprendizaje es usado comúnmente en modeladores funcionales y asociadores de patrones.

- **Aprendizaje no supervisado:** Se dispone únicamente de patrones de entrada [11]. Al no mostrar patrones objetivos, será la propia red quien decidirá si la salida es correcta o no. Mediante los patrones de entrada y los almacenados, la red va agrupando en categorías y clases los patrones de salida. En este caso los datos no están etiquetados.

  Hay dos tipos de aprendizaje no supervisado que podemos destacar:

  - **Aprendizaje por componentes principales:** Esta basado en obtener las características principales comunes a la mayoría de los patrones de entrada.
  - **Aprendizaje competitivo:** Consiste en seleccionar neuronas con los pesos más similares a los patrones de entrada. Posteriormente se reforzará la mejor neurona para que los pesos se asemejen a esta y se debilitarán las restantes.

- **Aprendizaje reforzado:** En este caso destaca la presencia de un supervisor el cual determina si la respuesta es válida o no. Se aprende tanto de los aciertos como de los fallos, pero cuando acierta se premian los pesos y cuando falla se penalizan.

## 2.7. Aprendizaje Auto-supervisado (Self-Supervised Learning)

Los sistemas automatizados de visión por computadora han producido avances enormes en diversas tareas. Actualmente muchos modelos que afrontan desafíos tales como detección, segmentación y reconocimiento de objetos pueden competir en rendimiento con los humanos en tareas específicas. Sin embargo, esto es gracias a la gran cantidad de datos etiquetados. El problema reside en que éstos, no siempre están disponibles y son costosos, además de que estos sistemas suelen estar diseñados y entrenados para tareas específicas y puede que no funcionen para otros escenarios.

Gracias a los avances recientes de investigación los sistemas pueden adaptarse a nuevas condiciones sin requerir una supervisión costosa. Esto ha influido positivamente en el aprendizaje auto-supervisado [12] [13].

El aprendizaje auto-supervisado [14] solo requiere datos sin etiquetar para formular las tareas pretexto. La idea principal es realizar una tarea pretexto como predecir la rotación de un conjunto de imágenes y gracias a esta tarea calcular el objetivo sin supervisión. Las capas

intermedias de las redes neuronales convolucionales son capaces de resolver este tipo de tareas que serán utilizadas para tareas futuras de interés como es la detección de imágenes.

Una ventaja natural del aprendizaje auto-supervisado respecto al aprendizaje supervisado es que los modelos al no necesitar una intervención humana para etiquetar los datos pueden actualizarse y ser entrenados completamente desde cero. Este aprendizaje es adecuado para entornos cambiantes.

Las técnicas auto-supervisadas aprenden de los conjuntos de datos. Estas han demostrado grandes resultados en aprendizaje de representaciones de imágenes de alto nivel. Hay diferentes formas de generar las etiquetas a utilizar en las imágenes. Además, hay diferentes métodos en las tareas de pretexto para el uso del aprendizaje auto-supervisado como pueden ser la coloración de imágenes o las restricciones estructurales en el espacio de representación.

El objetivo final es que los pesos aprendidos durante el aprendizaje auto-supervisado sirvan como puntos iniciales mejores que los aleatorios para el entrenamiento de la tarea objetivo. La idea principal de la tarea pretexto es que durante su aprendizaje se aprenden filtros caracterizadores que sirven para otras tareas.

### 2.7.1. Técnicas auto-supervisadas

- **Rotación (*Rotation*):** La tarea [15] funciona de tal forma que mediante el conjunto de datos se generan 4 copias con las imágenes giradas en {0, 90, 180, 270} grados (ver Figura 2.9). La red tiene que predecir la rotación de las imágenes aplicada donde un buen modelo debería aprender estas rotaciones en imágenes naturales.

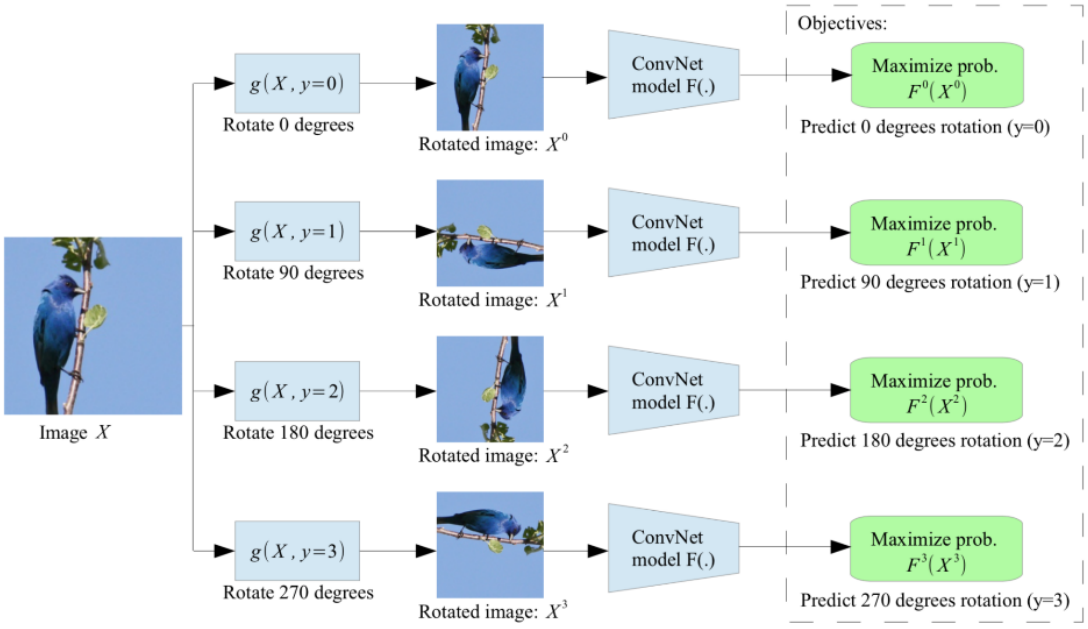



Figura 2.9: Ilustración del aprendizaje auto supervisado rotando todas las imágenes. El modelo predice la rotación aplicada. Extraído de [15].

- **Ejemplar (*Exemplar*):** En esta técnica [16], todas las imágenes están relacionadas con sus propias clases donde se generarán una multitud de ejemplos de estas a través de un aumento aleatorio grande de los datos (ver Figura 2.10). Tiene cierta similtud con el aumento de datos (*data-augmentation*), donde los datos son escalados, rotados, traducidos, con cambios de color y contraste.

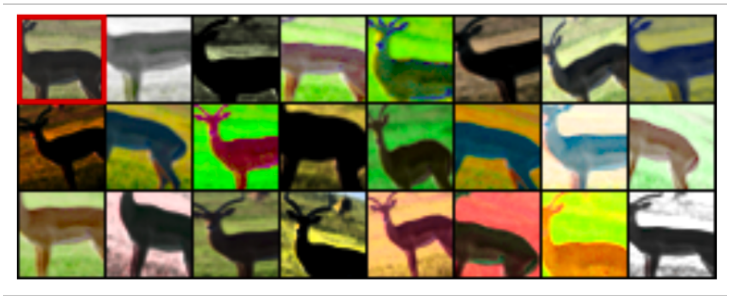


Figura 2.10: Técnica autosupervisada Exemplar. Se aplican transformaciones aleatorias lo que resulta en una variedad de parches distorsionados que pertenecen a la misma clase para la tarea pretexto. Extraído de [16].

- **Rompecabezas (*Jigsaw*):** La tarea pretexto [17] consiste en recuperar la posición espacial relativa de 9 parches de la imagen muestreados aleatoriamente después de una permutación al azar de estos parches (ver Figura 2.11). Los parches son enviados a través de la propia red neuronal, luego las representaciones se concatenan y pasan a través de un perceptrón multicapa que está totalmente conectado a las capas ocultas, donde predice la permutación utilizada. Posteriormente, se extraen las representaciones promediando las representaciones de 9 muestras normalizadas de los parches de la imagen.

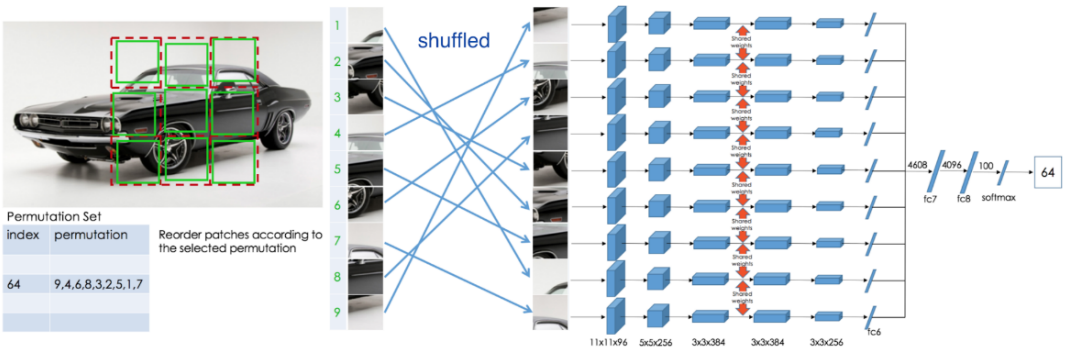



Figura 2.11: Ilustración del aprendizaje auto supervisado mediante la resolución rompecabezas. Extraído de [17].

- **Ubicación relativa del parche (*Relative Patch Location*):** Esta técnica [18] consiste en realizar una predicción de la ubicación relativa de dos parches situados en la imagen. Es similar a la anterior tarea pretexto mencionada (se utiliza el mismo parche) pero en este caso las posibles relaciones espaciales relativas entre los dos parches están limitadas a 8 posiciones (ver Figura 2.12).

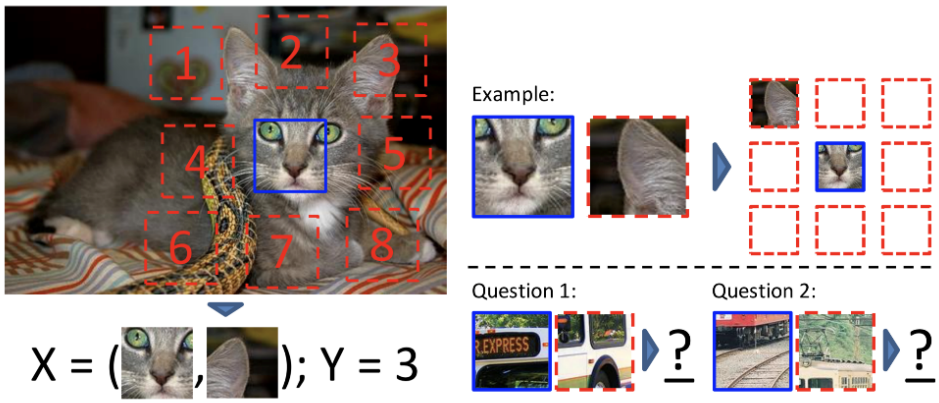



Figura 2.12: Ilustración del aprendizaje auto supervisado prediciendo la posición relativa de dos parches aleatorios. Extraído de [18].

- **Conteo de características (*Counting Features*):** La idea principal de esta tarea pretexto [19] es considerar como un valor escalar a las características o primitivas visuales que se resumen y comparan en diversos parches, para luego contar las características con la relación entre parches. Hay 2 transformaciones a considerar (ver Figura 2.13):

  La primera es la escala, donde si se duplica el tamaño de la imagen, el número de primitivas visuales debe permanecer igual.

  La segunda es el mosaico, en el cual si está en este formato, por ejemplo en una cuadricula de 2 x 2, el número de primitivas visuales se espera que sea la suma.

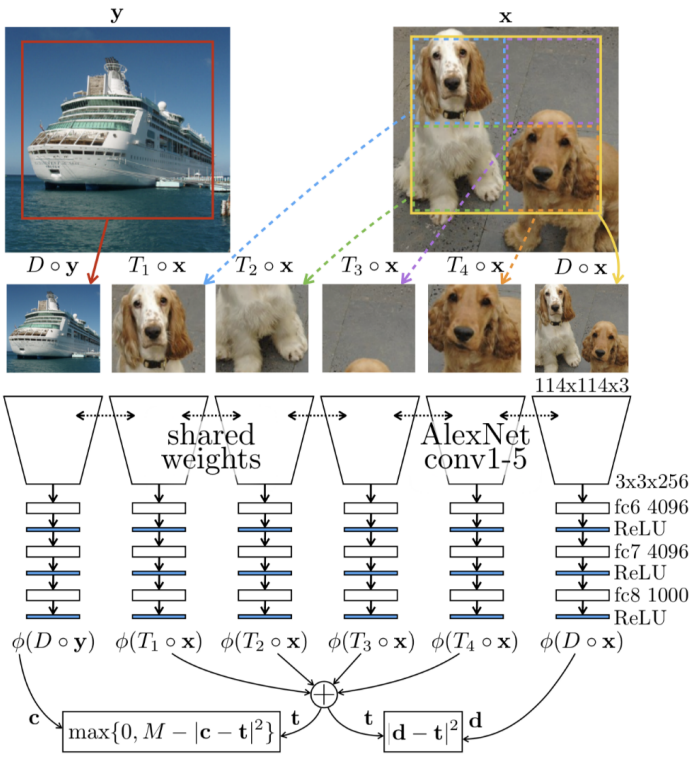



Figura 2.13: Representación del aprendizaje auto-supervisado por conteo de características. Extraído de [19].

- **Colorización (*Colorization*):** La colorización [20] resulta ser una tarea pretexto potente. La tarea consiste en mapear una imagen de entrada en escala de grises a una distribución sobre salidas en color cuantificadas.

  El modelo genera colores en el espacio Lab *. Este espacio de color está diseñado como una aproximación a la visión humana. El componente L* representa la percepción humana siendo el valor 0 negro y el valor 100 blanco. El componente a* coincide con el valor verde cuando es negativo y por el contrario magenta si es positivo. Por último, el componente b* modela el valor azul cuando es negativo y amarillo si es positivo.

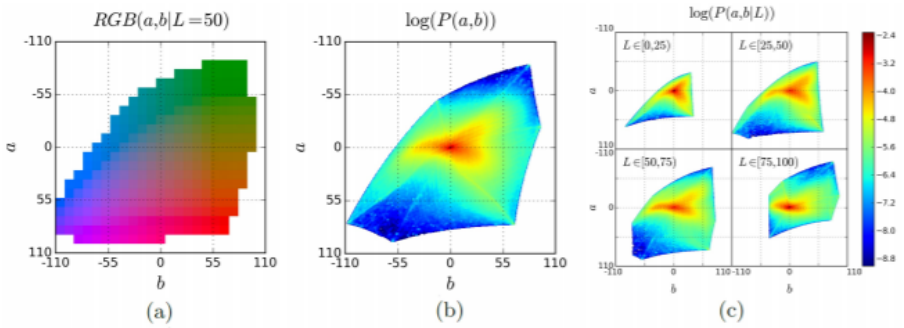



Figura 2.14: (a) y (b) son diferentes representaciones de colorización con valores ab. (c) depende del valor L. Extraído de [**?** ].

- **Codificación predictiva contrastante (*Contrastive Predictive Coding*):** La codificación predictiva contrastante (CPC) [21] traduce un problema con un modelo generativo a uno de clasificación. La pérdida de contraste en CPC se utiliza para medir qué tan bien puede clasificar el modelo la representación futura para un conjunto de muestras negativas no relacionadas.

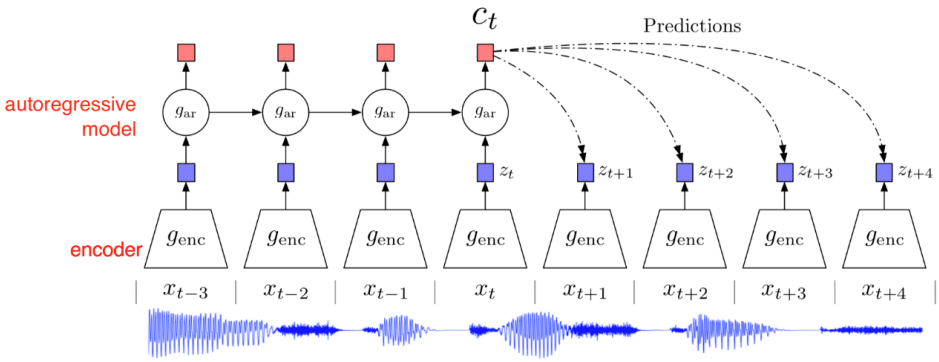



Figura 2.15: Representación de la técnica de codificación predictiva contrastante en audio. Extraído de [21].

La red de predicciones solo accede a un conjunto de características enmascaradas, evitando así una predicción trivial. Cada imagen se divide en varios parches superpuestos codificados por la red resultando en un vector de características comprimido (ver Figura 2.16).

La predicción es realizada por una red de comunicación enmascarada a través de una máscara donde el campo receptivo de una neurona de salida solo puede ver cosas sobre ella en la imagen. Esta predicción puede ser realizada tanto de arriba hacia abajo como de abajo hacia arriba.

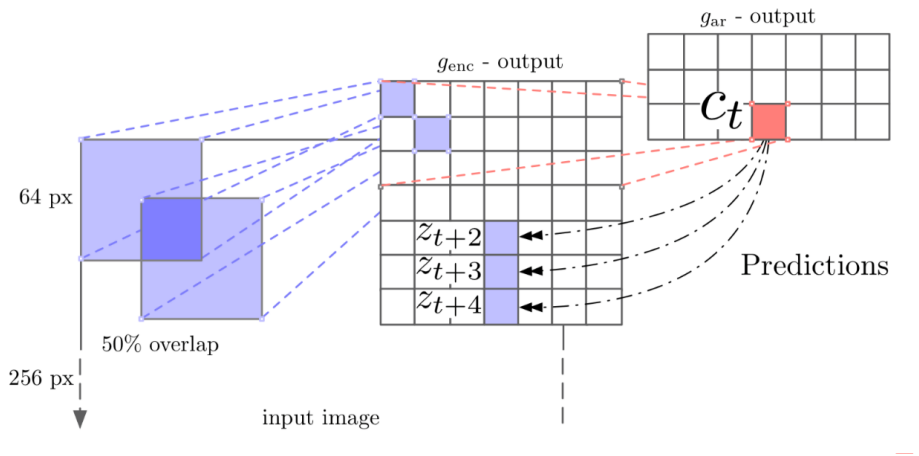



Figura 2.16: Representación de la técnica de codificación predictiva contrastante en imágenes. Extraído de [21].

- **Contraste de momento (*Momentum contrast*):** Proporciona una búsqueda dinámica de diccionario estructurado como una cola FIFO con las representaciones codificadas de muestras de datos [22].

  Dada una muestra, se pasa por un codificador y se obtiene una representación. Por otra parte, tenemos muestras clave que son codificadas por un codificador de impulso para producir una lista de representaciones clave. Luego se calcula la pérdida de contraste para las muestras positivas y claves negativas (ver Figura 2.17).

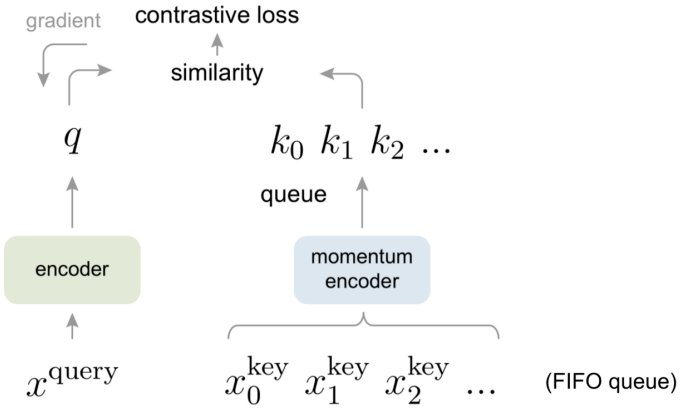



Figura 2.17: Ilustración de cómo Momentum Contrast aprende representaciones visuales. Extraído de [22].

# 3

# Diseño y Desarrollo

## 3.1. Introducción

Tras haber analizado el estado del arte se describen brevemente los entornos de desarrollo que se han utilizado para el desarrollo del trabajo. Por otro lado, se verán las bases de datos utilizadas en los experimentos, así como la arquitectura de la red neuronal utilizada de forma general y también dicha arquitectura, pero ajustada a ambas bases de datos. Además, se hará una explicación de las técnicas usadas para el desarrollo, *Transfer Learning* y *Fine Tuning*. Por último, se hará una explicación acerca del desarrollo que se ha realizado para poder obtener los resultados y la tarea pretexto elegida.

## 3.2. Entornos de desarrollo

### 3.2.1. Matlab

Matlab es una plataforma de programación basada en lenguaje de matrices y que está optimizada para resolver problemas científicos.

Destaca el paquete de Deep Learning (*Deep Learning ToolBox*), el cual nos permite implementar y diseñar redes neuronales además de una cantidad enorme de funciones y redes previamente entrenadas.

El entorno de Matlab ha sido utilizado para realizar las primeras pruebas del trabajo realizado donde se examinó la primera base de datos y se realizaron experimentos para posteriormente llevarlos a cabo en Pytorch de una forma más simple con los conocimientos obtenidos.

### 3.2.2. Pytorch

Pytorch [23] es una librería muy reciente que fue lanzada en 2017. Esta plataforma dispone de una cantidad enorme de tutoriales acerca de todas sus implementaciones. Implementa una interfaz muy sencilla para la creación y manipulación de redes neuronales, aunque Pytorch trabaja con tensores directamente por lo que no necesita de librerías a niveles superiores como hacen otros entornos.

Otra de las ventajas de Pytorch es que tiene soporte para ejecutar en tarjetas gráficas (GPU). En concreto utiliza CUDA, una API desarrollada por NVIDIA que conecta la CPU con la GPU para disminuir los tiempos de ejecución drásticamente.

Además, Pytorch respecto a otros paquetes trabaja con grafos dinámicos en vez de estáticos. Esto es que mientras se está ejecutando el programa, se pueden modificar los cálculos y las operaciones.

#### Google Colaboratory

Google Colaboratory (Google Colab), es un entorno que permite programar y ejecutar código Python en tu navegador. Los cuadernos te permiten hacer combinaciones tanto de código a ejecutar como de texto informativo en el mismo documento, además de poder añadir imágenes.

Lo más importante de este entorno es que te permite conectar los cuadernos del Google Colab con tu cuenta de Google Drive. Así puedes cargar conjuntos de datos guardados en el Drive, importar códigos de GitHub y otras fuentes.

Los cuadernos de Google Colab aprovechan la potencia del hardware de Google donde incluyen tanto GPU como TPU. Al estar conectado a la nube, se pueden entrenar redes neuronales con GPU para acelerar los procesos sin tener en cuenta la potencia del equipo que se disponga.

Tanto Pytorch como Google Colab fueron elegidos por el gran aumento de velocidad que suponía al permitir GPU. Por otra parte, al ser un trabajo basado en clasificación de imágenes, estas plataformas presentan una gran cantidad de ventajas y funcionalidades comentadas anteriormente.

## 3.3. Bases de datos

### 3.3.1. Clasificación en dominio convencional

El primer dataset utilizado es CIFAR10 [24]. Este conjunto de datos consta de 60000 imágenes coloreadas (RGB) de tamaño 32 x 32 píxeles en formato png con 10 tipos de clases distintas (6000 imágenes por clase).

El número de imágenes está dividido en 50000 imágenes para el conjunto de entrenamiento (train) y 10000 imágenes para el conjunto de prueba (test).

Este dataset se ha utilizado como base para posteriormente hacer los experimentos con el dataset de imágenes no convencionales.

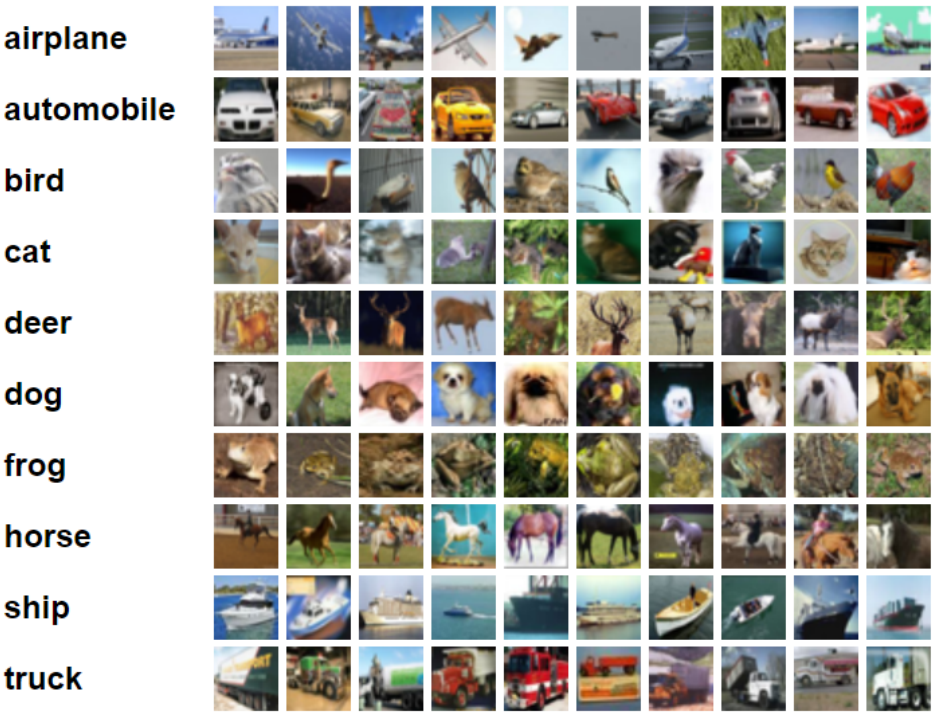



Figura 3.1: 10 imágenes aleatorias de cada clase del conjunto de datos CIFAR10. Extraído de [24].

### 3.3.2. Clasificación en dominio no convencional

El segundo dataset utilizado es el proporcionado por la competición ISIC 2019, *Skin Lesion Analysis Towards Melanoma Detection* [25] [26]. En concreto, el conjunto de datos proviene de *BCN20000* [27], que son imágenes dermatoscópicas de las clases más comunes de lesiones cutáneas. Hay 8 categorías distintas en las que englobar las imágenes:

El conjunto de datos consta de 25331 imágenes coloreadas (RGB) con un tamaño de 1024 x 1024 píxeles en formato jpg. Las imágenes están divididas en 8 estados diferentes de lesiones de piel.

Para dividir las imágenes en un conjunto de entrenamiento y un conjunto de prueba, hemos elegido un porcentaje similar al que teníamos con la primera base de datos. El sistema anterior presenta un 67 % de las imágenes en el conjunto de entrenamiento y un 33 % en el de prueba. En esta ocasión hemos elegido un 70 % de las imágenes para el conjunto de entrenamiento y un 30 % para el conjunto de prueba.

Antes de pasar al siguiente punto, se representan en una tabla el número de imágenes procedentes de cada categoría y el total de ellas. Tanto para el conjunto de entrenamiento como para el de prueba.

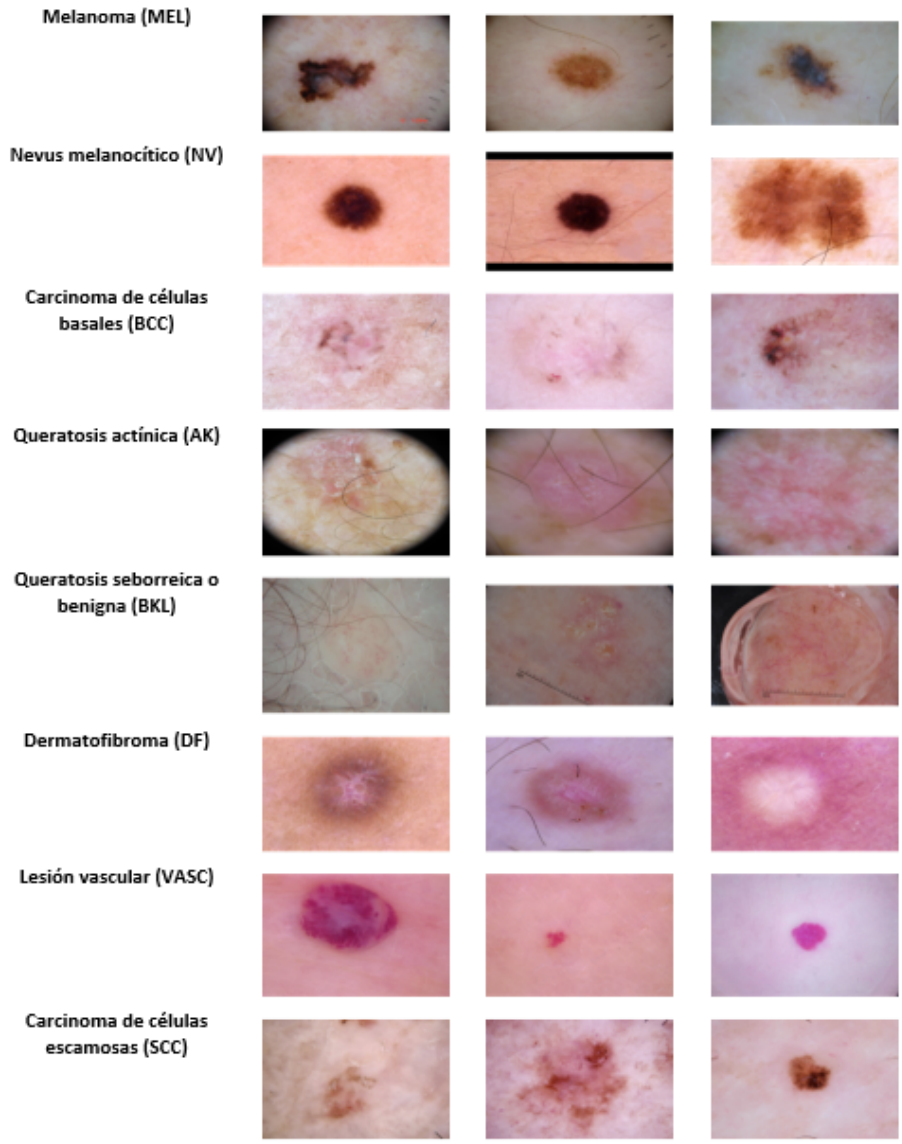



Figura 3.2: 3 imágenes aleatorias de cada clase del conjunto de datos ISIC 2019.

| CLASES | TRAIN | TEST |
|---|---|---|
| **MEL** | 2716 | 1806 |
| **NV** | 10169 | 2706 |
| **BCC** | 1527 | 1796 |
| **AK** | 391 | 476 |
| **BKL** | 1875 | 749 |
| **DF** | 160 | 79 |
| **VASC** | 175 | 78 |
| **SCC** | 359 | 269 |
| **TOTAL** | 17372 | 7959 |

Tabla 3.1: Número de imágenes por clase y totales del dataset completo ISIC 2019.

## 3.4. Arquitectura de la CNN

### 3.4.1. ResNet (*Residual neural network*)

La creación de este tipo de arquitectura en las redes neuronales convolucionales nace de la siguiente observación: Las CNN cuando se le añadían más capas y eran más profundas no sólo no mejoraban la precisión, sino que a veces incluso la empeoraban. Motivados por intentar conseguir una mejora en la red, en vez de apilar más capas una tras otra, implementaron los bloques residuales donde las capas adyacentes aprenden características de mapas residuales como se muestra en la **Figura 3.3**.

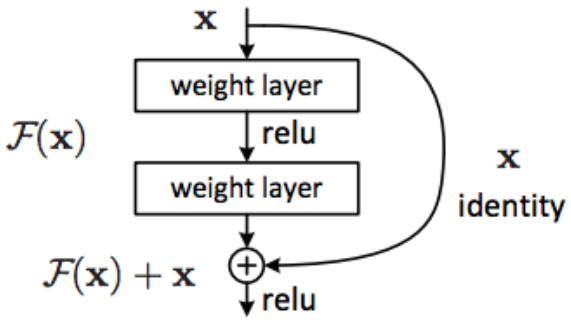



Figura 3.3: Bloque residual. La información se actualiza con la función residual o pasa directamente por la identidad. Extraído de [28].

De esta forma, se puede transmitir información importante de capas anteriores a capas posteriores. Las capas de estos bloques aprenden las diferencias entre la entrada y la salida. Por tanto, dichos bloques permiten añadir más capas a la red sin tener una repercusión negativa ya que el mapa de activación que llega a la entrada será el óptimo evitando así el problema de desvanecimiento de gradiente.

Si la entrada no es óptima el gradiente pasará por la identidad. En caso contrario, esté se actualizará comparándose con la función residual F(x) sin tener que rehacer el mapa.

Además, estos bloques aceleran el entrenamiento de la red.

A continuación, se muestra una comparativa entre tres tipos de redes. Como se puede ver en la **Figura 3.4**, la tercera red es una ResNet con 34 capas en la cual hay varias capas convolucionales residuales. Se puede observar una capa *average pooling* antes de llegar al final para reducir la salida. Por último, tenemos una capa clasificadora *fully connected* que en este caso está adaptada a 1000 clases (Adaptada al conjunto de datos de Imagenet que usa 1000 clases).

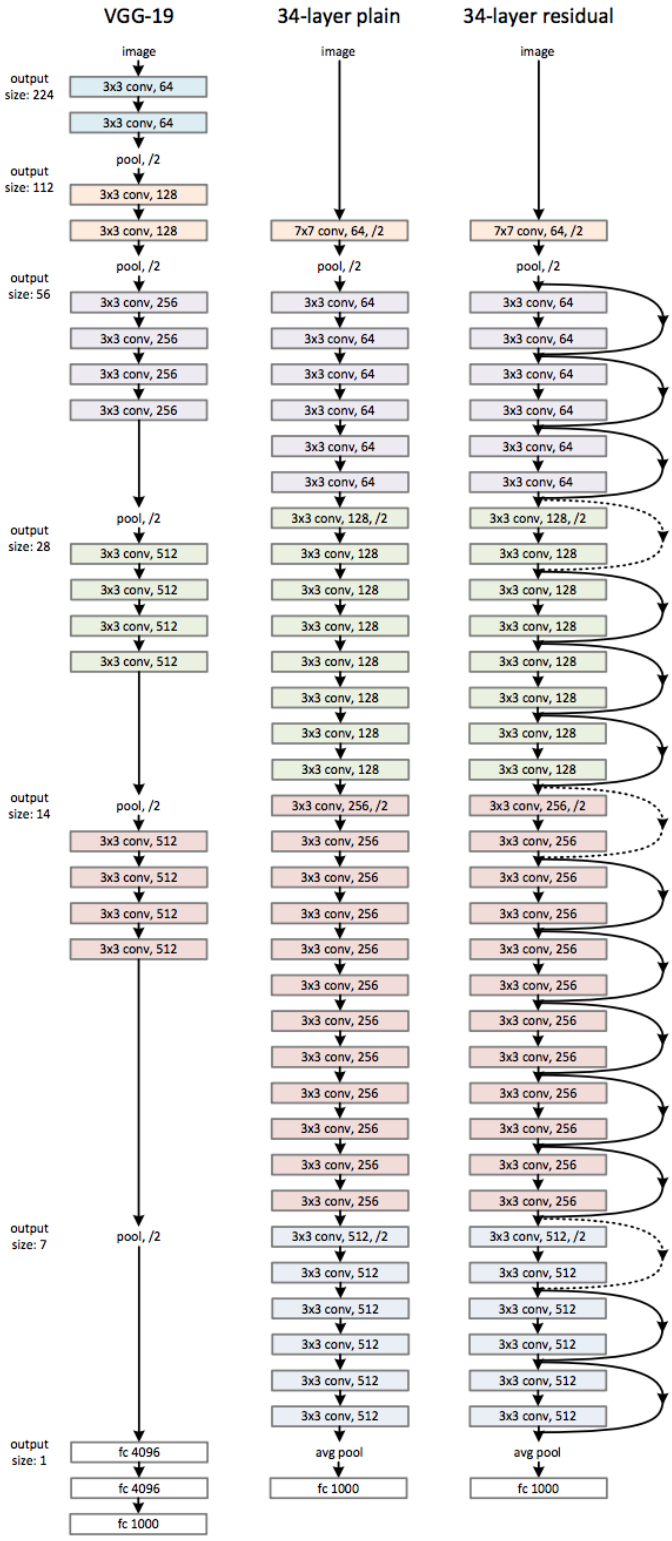



Figura 3.4: Ejemplos de arquitecturas de red. A la derecha una red residual de 34 capas de parámetros. Los atajos de puntos aumentan las dimensiones. Extraído de [28].

Como cualquier tipo de red neuronal, las redes neuronales residuales se pueden modificar. A continuación, se mostrarán las dos redes residuales utilizadas para cada conjunto de datos.

### Resnet ajustada al primer *dataset*

Para el primer dataset se utiliza una Resnet de 32 capas que está organizada de la siguiente manera:

1. **Conv2d:** Una capa convolucional 2D (2 dimensiones) con 3 canales de entrada, 16 canales de salida, kernel de 3 x 3, stride de 1 x 1 y un padding de 1 x 1.
2. **BatchNorm2d:** Una capa de normalización por lotes sobre una entrada 4D (minilote de entradas 2D). Como la salida de la capa convolucional presenta 16 canales, esta capa en la entrada tiene 16 canales también.
3. **Layer:** Esta capa es modificada manualmente. Tenemos 3 capas (*layers*) las cuales se componen cada una de un secuencial con 5 bloques. Cada bloque esta formado por una capa convolucional, seguida de una capa de normalización, luego otra capa convolucional y por último otra capa de normalización. Lo que destaca de estas 3 capas es que progresivamente aumentamos el número de entradas y salidas. En la primera capa tendremos 16 canales de entrada y salida, en la segunda capa 32 y en la tercera y última 64.
4. **Linear:** Es la capa final de la red la cual aplica una transformación lineal a los datos de entrada. Presenta los 64 canales de la salida anterior en la capa de entrada y de salida el número de clases, que en este caso al trabajar con el *dataset* de CIFAR10 son 10.

Lo interesante de este modelo es que no hay capa de reducción (*pooling*). Esto es debido a que las imágenes de CIFAR10 son muy pequeñas (32 x 32) y por tanto no es conveniente reducir el tamaño de las imágenes ya que haría que el modelo no fuera efectivo para este conjunto de datos.

### Resnet ajustada al segundo *dataset*

Para el segundo conjunto de datos utilizamos una Resnet de 18 capas pero que tiene variaciones con respecto a la mencionada anteriormente:

1. **Conv2d:** Capa convolucional con 3 canales de entrada como en el anterior modelo. La salida esta vez tiene 64 canales y al ser imágenes más grandes aumentamos el kernel a 7 x 7, el stride a 2 x 2 y el padding a 3 x 3.
2. **BatchNorm2d:** Capa de normalización con 64 canales de entrada igual al primer modelo.
3. **ReLU:** Función de activación ReLU.
4. **MaxPool2d:** Capa de reducción basada en coger los máximos.
5. **Layer:** Esta vez tenemos 4 capas, pero distribuidas de manera diferente. Cada capa tiene un secuencial pero con 2 bloques en vez de 5 como en el anterior modelo. Cada bloque está formado internamente igual que el anterior modelo, pero entre la primera capa de normalización y la segunda capa convolucional se introduce una ReLU. Además, las capas convolucionales tendrán más entradas y salidas. En la primera capa 64, en la segunda 128, en la tercera 256 y en la cuarta 512.
6. **AdaptiveAvgPool2d:** Capa de reducción que aplica una agrupación promedio sobre la señal de entrada.

7. **Linear:** Con 512 canales de entrada. En este modelo tenemos la salida será de 8 canales ya que el conjunto de datos ISIC 2019 tiene 8 clases.

Al tener pocas imágenes en nuestro dataset, se introduce manualmente una capa **Dropout** la cual desactiva un número de neuronas de forma aleatoria. Esta capa ayuda al aprendizaje de la red y a evitar el overfitting. En nuestro modelo la hemos añadido con un valor de 0.5 que es el que se suele usar por defecto: la mitad de las neuronas estarán activas y la otra mitad no.

## 3.5. Transfer Learning y Fine-Tuning

En la práctica, muy pocas veces se entrena toda la CNN desde cero con inicialización de pesos aleatoria, debido a que usualmente no se posee de un conjunto de datos de tamaño suficiente. Por otra parte, es muy común pre-entrenar una CNN con un conjunto de datos grande y luego utilizar esa CNN como extractor de características o usarla de inicialización para posteriores tareas. Esto es lo que usamos en nuestro trabajo con la tarea pretexto. Hay 3 escenarios importantes dentro del Transfer Learning [29] [30] [31]:

- **CNN como extractor de características fijas:** Se toma la CNN previamente entrenada y se congelan todas las capas de la red excepto la última (clasificador totalmente conectado, *fully connected*) que se sustituye por otra igual, pero con pesos aleatorios y se entrena.
- **Fine-Tuning:** En esta estrategia no solo se reemplaza el clasificador, sino también se ajustan los pesos de la red pre-entrenada al continuar con el *backpropagation*. Se pueden ajustar tanto todas las capas de la CNN como dejar algunas fijas y solo ajustar las capas superiores. Esto viene motivado porque las características de las primeras capas de las CNN suelen responder a características más generales (detección de bordes) y progresivamente las capas posteriores se vuelven más específicas.
- **Modelos pre-entrenados:** Debido a que las CNN actuales pueden llegar a tardar meses en entrenar incluso con GPU, las personas comparten sus puntos de control para un beneficio común.

Esta técnica es muy útil cuando varía el tamaño del nuevo conjunto de datos y la similitud con el conjunto de datos original.

En nuestro caso, el nuevo conjunto de datos es pequeño, pero similar al conjunto de datos original, ya que la tarea pretexto que usamos es la rotación y por tanto son las mismas imágenes (mismo tamaño) pero giradas.

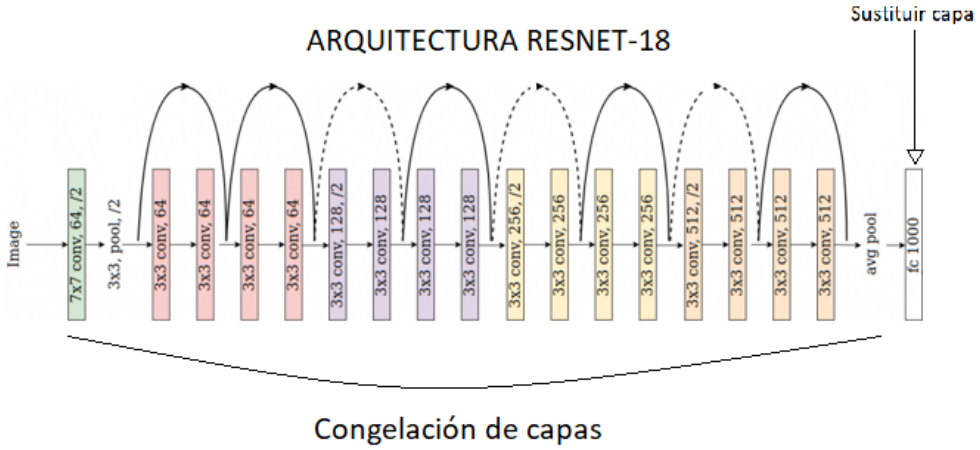



Figura 3.5: Transfer learning en la arquitectura usada para el segundo conjunto de datos. Se congelan todas las capas y se sustituye el clasificador cambiando el número de clases que requiera. Poco a poco se van descongelando las capas.

## 3.6. Desarrollo base

En primer lugar, se importan las librerías necesarias para el entrenamiento (*numpy, torchvision*). Posteriormente se definen los hiperparámetros como el número de épocas, el *learning rate*, etc. Se declaran las rutas de las carpetas en las que se encuentran nuestros datos y donde vamos a guardar los resultados. Se carga nuestro modelo de red y se configura para utilizar GPU.

**NOTA:** En este punto se congelarán las capas si se requiere para que únicamente pasen a la fase de entrenamiento las capas deseadas.

Se cargan los datasets de entrenamiento y de prueba donde las imágenes están separadas por carpetas de *train* y *test* y dentro de estas subcarpetas con cada clase. En esta fase se realizan las transformaciones a las imágenes de redimensión, normalización y de pasarlas a formato Tensor ya que Pytorch trabaja con este tipo de datos. También se hace aplica el *data augmentation* el cual genera más datos de entrenamiento a partir de los que tenemos con el objetivo de generalizar mejor y ayudar en el aprendizaje. Estos datos extras son generados a partir de una serie de transformaciones a las imágenes como adicción de ruido gaussiano a las imágenes, cambio de contraste, aumentar y disminuir el brillo, etc.

Antes de pasar al entrenamiento se decide que queremos predecir. En nuestro caso dependiendo del paso de los experimentos donde nos encontremos se decidirá entre predecir la rotación (tarea pretexto) o clasificar las imágenes (tarea base).

A continuación, se entrena y se valida el modelo. Estas funciones se definen dentro de un bucle que va de época en época calculando la precisión y la pérdida de los resultados obtenidos. Estos datos se guardan en un punto de control (*checkpoint*) el cual será usado para las tareas de *transfer learning*.

### 3.6.1. Tarea Pretexto

Tras haber analizado las diferentes técnicas auto-supervisadas en la Sección 2.7.1, nos hemos decantado por elegir la de rotación. Para tareas en entornos RGB la técnica de rotación es la que suele producir mejores resultados, ya que suele ser más fácil de predecir correctamente.

Después de realizar las transformaciones e implementar el *data augmentation* mencionados anteriormente, se realiza la tarea pretexto (en nuestro caso rotación) cuando sea necesario sacar los resultados con la misma.

Una vez realizada la prueba con la tarea pretexto, se guarda nuestro mejor modelo para posteriormente en los siguientes pasos partir de él. De esta forma, al realizar la clasificación de imágenes partiendo de un modelo entrenado para rotaciones, asumimos que le resultará más facil a la red aprender características de las muestras ya que habrá aprendido de la tarea pretexto.

**NOTA:** Destacar que en la mayoría de modelos, se suelen utilizar tanto rotaciones fijas como aleatorias en las transformaciones y *data augmentation*, las cuales han sido eliminadas de nuestro modelo ya que si estas transformaciones existieran, nuestro modelo no aprendería adecuadamente de la tarea pretexto de rotación.

# 4
# Evaluación

## 4.1. Introducción

Una vez visto el desarrollo de nuestro sistema (bases de datos, arquitectura de la red, técnicas y algoritmo base), se procederá en este capítulo a realizar y evaluar las pruebas. En primer lugar, vamos a definir los pasos que se deben seguir para realizar las pruebas. A su vez, propondremos un objetivo que marcará la finalidad de los experimentos. Posteriormente, se efectuarán las pruebas y se analizarán brevemente los resultados obtenidos gráficamente. Por último, se hará un análisis y comparativa entre los resultados obtenidos a través de dos tablas generales que abarcan todo el proceso de pruebas.

## 4.2. Pasos y Objetivo

En primer lugar, se definen los 3 conjuntos de datos diferentes que se emplearán para realizar las pruebas:

- **Set_1_T:** Conjunto de datos de entrenamiento (train) completo. Contiene todas las imágenes de entrenamiento.
- **Set_1_V:** Conjunto de datos de prueba (test) completo. Contiene todas las imágenes de prueba.
- **Set_2_T:** Conjunto de datos de entrenamiento (train) reducido. Contiene un 10 % de las imágenes que tiene el Set_1_T.

A continuación, se definen los pasos que se efectúan en las pruebas:

**Paso 1.** Entrenar un modelo de clasificación con Set_1_T y validar con Set_1_V. Referencia completa → *Resultado*1.

**Paso 2.** Entrenar un modelo de clasificación con Set_2_T y validar con Set_1_V para elegir la mejor época. Referencia subset → *Resultado*2.

**Paso 3.** Entrenar un modelo con la tarea pretexto con Set_2_T y validar dicha tarea pretexto con Set_1_V para elegir la mejor época. Referencia pretexto → $Modelo\ auto-supervisado$.

**Paso 4.** Ajustar a un modelo de clasificación el modelo auto supervisado (congelando capas) con Set_2_T y validar con Set_1_V para elegir la mejor época. Referencia pretexto + subset → *Resultado*4.

Este paso se dividirá en 5 sub-pasos:

- Se congelarán todas las capas menos la última (el clasificador).
- Se congelará el 75 % de las capas.
- Se congelará el 50 % de las capas.
- Se congelará el 25 % de las capas.
- No se congelará ninguna capa.

El objetivo que se desea lograr es el siguiente: Conseguir que el Resultado 4 sea mejor que el Resultado 2.

## 4.3. Pruebas y Resultados

### 4.3.1. Resultados sobre dominios convencionales

En las siguientes pruebas realizadas, se utiliza un LR (*learning rate*) variante durante 200 épocas. De las épocas 0-99, el LR es 0.1. De las épocas 100-149, el LR es 0.01. Por último, de las épocas 150-199 el LR es 0.001.

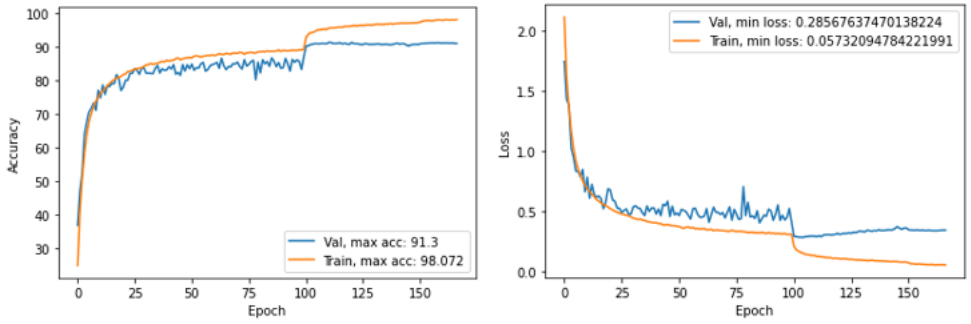
**Paso 1.** Como primer experimento, se emplea el 100 % de los datos para ambos conjuntos (entrenamiento y prueba).

**Paso 2.** En segundo lugar, se realizan las pruebas de forma similar al paso anterior pero cambiando el número de datos del conjunto de entrenamiento a un 10 %.

A partir de este punto, todos las pruebas posteriores son realizadas con un 10 % de los datos para el conjunto de entrenamiento.

A continuación, mostramos gráficamente los resultados obtenidos, representando la precisión (*accuracy*) y la pérdida (*loss*):

- Paso 1




- Paso 2

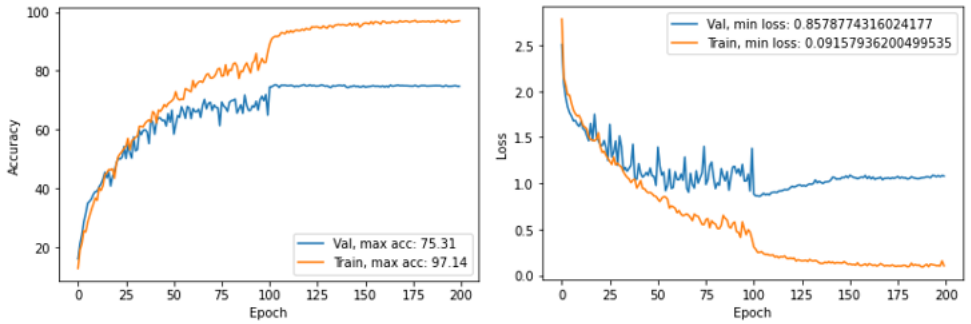



Figura 4.1: Resultado gráfico en 200 épocas representando la precisión y pérdida de los pasos 1 y 2 empleando el conjunto de datos de CIFAR10.

Como se observa, los resultados del Paso 1 son bastante buenos llegando a tener más de un 90 % de precisión. En el Paso 2 como es de esperar, los resultados empeoran con respecto al experimento anterior al tener menos imágenes de entrenamiento, el modelo aprende menos características de las muestras.

**Paso 3.** Se realizan las pruebas para la tarea pretexto. Se parte de las imágenes anteriores, las cuales son rotadas aleatoriamente de 4 formas distintas {0.º, 90.º, 180.º, 270.º}.

En este punto se realiza la parte del aprendizaje no supervisado (datos no etiquetados).

- Paso 3

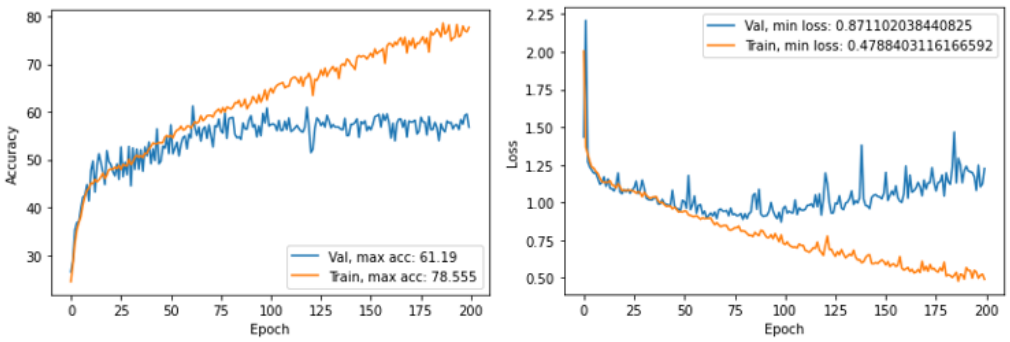



Figura 4.2: Resultado gráfico en 200 épocas representando la precisión y pérdida de la tarea pretexto (Paso 3) empleando el conjunto de datos de CIFAR10.

Empeoran los resultados con respecto a los anteriores pasos, aunque se tratan de tareas distintas no comparables. Además, hemos de tener en cuenta que se sigue utilizando un 10 % de los datos del conjunto de entrenamiento. En esta ocasión los datos no están etiquetados y debido a que las imágenes son muy pequeñas (tienen pocos píxeles), el modelo no puede aprender más de la tarea de rotación efectuada en las imágenes.

**Paso 4.** Por último, se procede a congelar todas las capas de la red tras el entrenamiento de la tarea pretexto y a ir descongelando en porcentaje para ver como mejoran los resultados y comprobar la viabilidad del objetivo.

- Paso 4_1 (Última capa sin congelar)

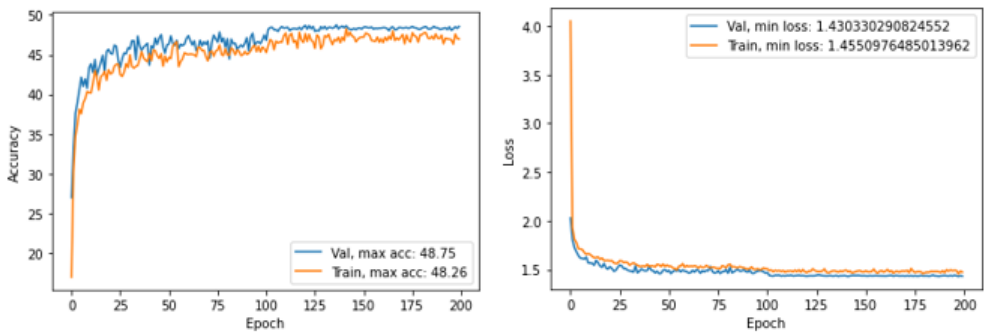



Figura 4.3: Resultado gráfico en 200 épocas representando la precisión y pérdida del Paso 4_1 empleando el conjunto de datos de CIFAR10.

- Paso 4_2 (75% congelado)

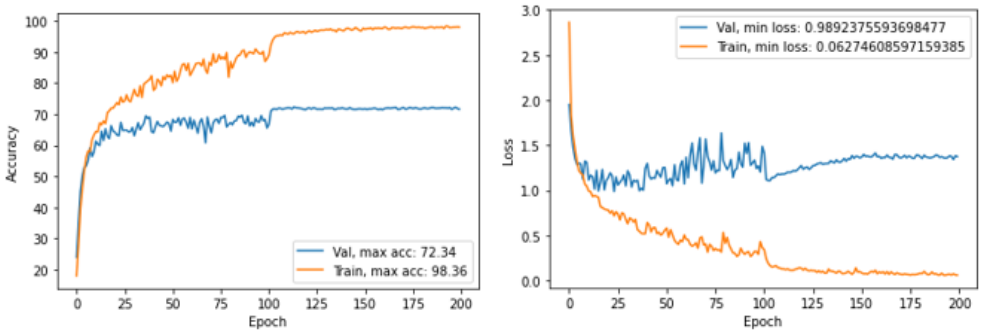



Figura 4.4: Resultado gráfico en 200 épocas representando la precisión y pérdida del Paso 4_2 empleando el conjunto de datos de CIFAR10.

- Paso 4_3 (50% congelado)

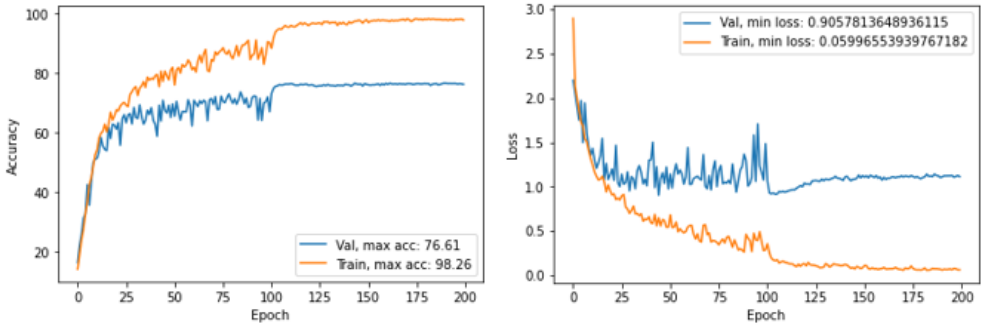



Figura 4.5: Resultado gráfico en 200 épocas representando la precisión y pérdida del Paso 4_3 empleando el conjunto de datos de CIFAR10.

- Paso 4_4 (25% congelado)

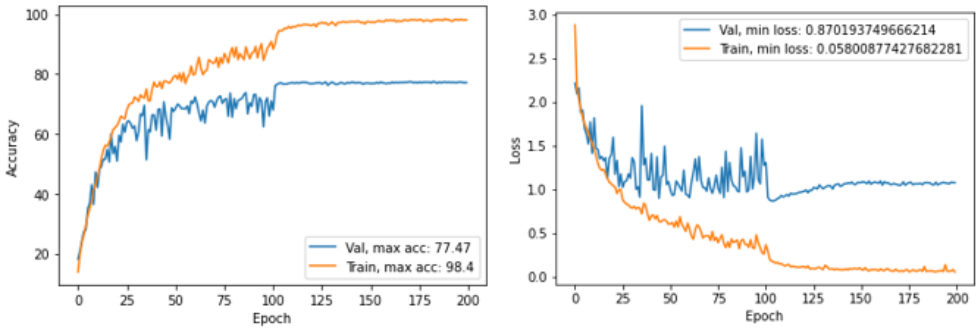



Figura 4.6: Resultado gráfico en 200 épocas representando la precisión y pérdida del Paso 4_4 empleando el conjunto de datos de CIFAR10.

- Paso 4_5 (Todo descongelado)

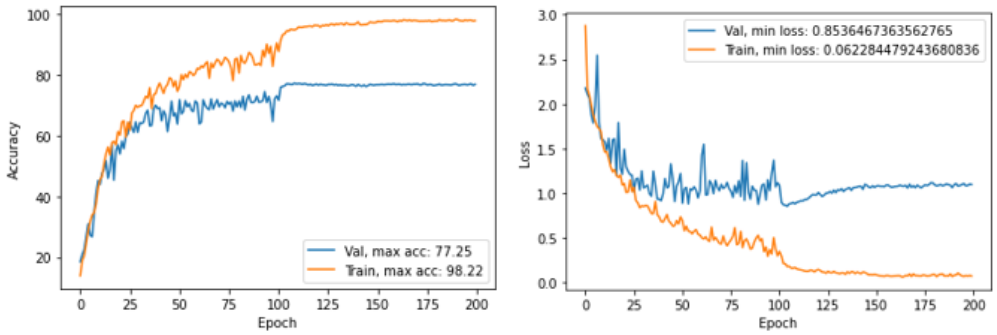



Figura 4.7: Resultado gráfico en 200 épocas representando la precisión y pérdida del Paso 4_5 empleando el conjunto de datos de CIFAR10.

Como se observa en los resultados (gráficas), a partir de tener un 75 % de las capas congeladas, el modelo empieza a tener resultados similares al Paso 2. De hecho, desde el Paso 4_3 (50 % de las capas congeladas), el modelo mejora con respecto al Paso 2 el cual era nuestro

objetivo a evaluar.

Los resultados sugieren que el efecto de la tarea pretexto cada vez tiene más peso según se descongela, porque las características aprendidas no son de utilidad para la tarea objetivo directamente, pero al ir descongelando se *adaptan* a ésta, y son preferibles al uso de pesos aleatorios para inicializar el modelo (tales como los usados en el Paso 2). En cualquier caso, el rendimiento final está aún lejos (77.47 % vs 91.3 %) de los obtenidos al usar el *dataset* completo (Paso 1).

### 4.3.2. Resultados sobre dominios no convencionales

En esta ocasión, contamos con el conjunto de datos de imágenes no convencionales. Al realizar las primeras pruebas nos dimos cuenta que gráficamente los resultados no eran correctos ya que como se puede ver en la figura la curva de entrenamiento comenzaba directamente desde un 60 % aproximadamente. Además, las dos curvas están muy separadas desde el comienzo lo que indica que algo fuera de lo común está sucediendo.

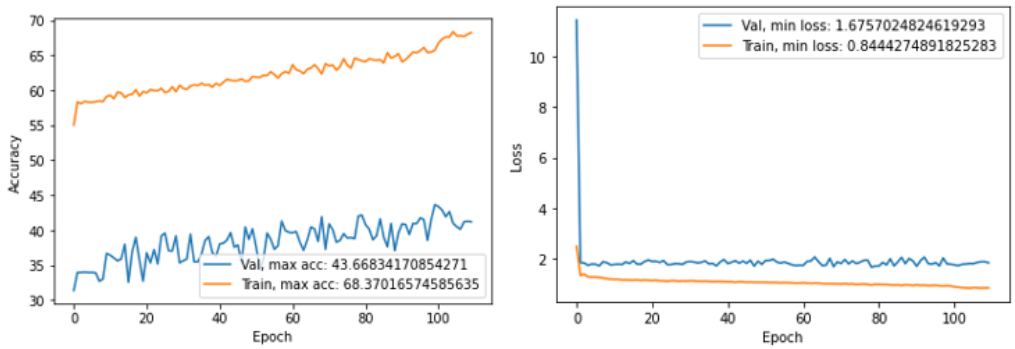



Figura 4.8: Resultado gráfico en 200 épocas representando la precisión y pérdida del paso 1 empleando el conjunto de datos de ISIC 2019. (Antes de modificar el conjunto de datos).

Revisando la Tabla 3.1 se observa que hay un desbalanceo de datos. Antes de realizar las pruebas balanceamos el dataset reduciéndolo a 800 imágenes tanto para el conjunto de entrenamiento como para el de prueba (100 imágenes por clase) (ver Tabla 4.1). También, otro importante factor para reducir las imágenes es su tamaño, ya que la herramienta de Google Colaboratory no soportaba tales dimensiones de las imágenes (1024 x 1024).

| CLASES | TRAIN | TEST |
|---|---|---|
| **MEL** | 100 | 100 |
| **NV** | 100 | 100 |
| **BCC** | 100 | 100 |
| **AK** | 100 | 100 |
| **BKL** | 100 | 100 |
| **DF** | 100 | 100 |
| **VASC** | 100 | 100 |
| **SCC** | 100 | 100 |
| **TOTAL** | 800 | 800 |

Tabla 4.1: Número de imágenes por clase y totales del dataset ISIC 2019 utilizado para las pruebas.

En las siguientes pruebas realizadas, se utiliza un LR (*learning rate*) fijo durante 200 épocas. El valor del LR durante el entrenamiento es de 0.1.

**Paso 1 y 2.** Una vez balanceado nuestro conjunto de datos, se dispone ha realizar las pruebas. Como hemos mencionado anteriormente, para el Paso 1 utilizaremos todos lo datos y para el Paso 2 únicamente el 10 % de las muestras del conjunto de entrenamiento.

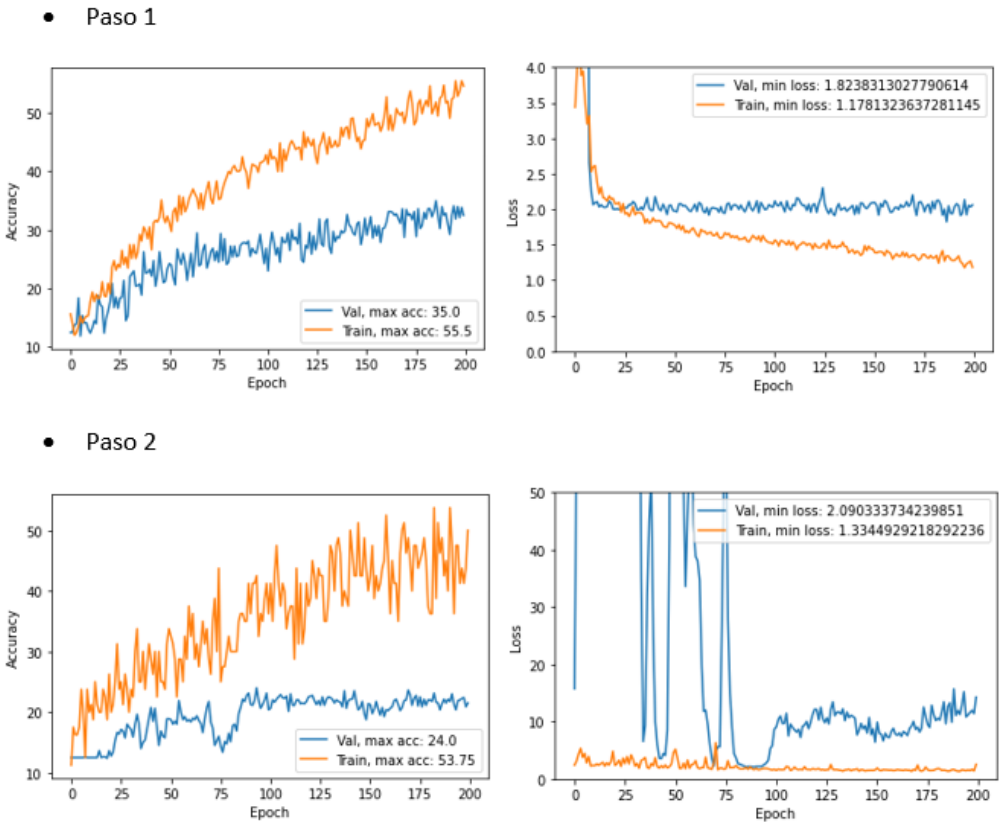



Figura 4.9: Resultado gráfico en 200 épocas representando la precisión y pérdida de los pasos 1 y 2 empleando el conjunto de datos de ISIC 2019.

Los resultados no son buenos debido a la gran reducción de imágenes en nuestro conjunto de datos. Además, el Paso 2 presenta un 10 % de las imágenes de entrenamiento y por ello es normal que la red no realice un aprendizaje efectivo. Aún así, el objetivo de este TFG no es clasificar imágenes de lesiones de piel excelentemente, sino ver si el efecto de aprendizaje auto-supervisado es útil en este tipo de dominios.

**Paso 3.** A continuación, se realiza la prueba con la tarea auto-supervisada que sigue siendo la rotación de imágenes en 4 ángulos distintos.

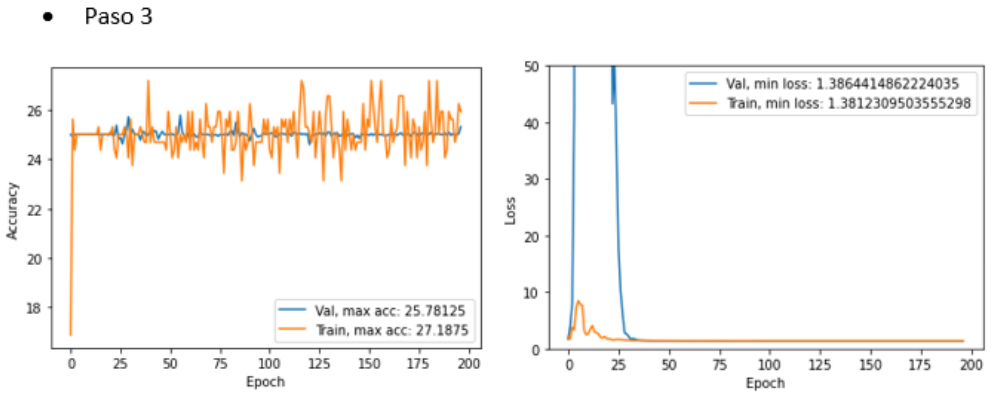



Figura 4.10: Resultado gráfico en 200 épocas representando la precisión y pérdida de la tarea pretexto (Paso 3) empleando el conjunto de datos de ISIC 2019.

Como se ve en la **Figura** 4.10 la máxima precisión alcanzada ronda los 25 %. Esto parece indicar que este tipo de imágenes no son variantes a rotaciones ya que la red siempre predice la misma rotación.

**Paso 4.** A continuación se congelan todas las capas. Poco a poco se descongelan las capas de la red progresivamente.

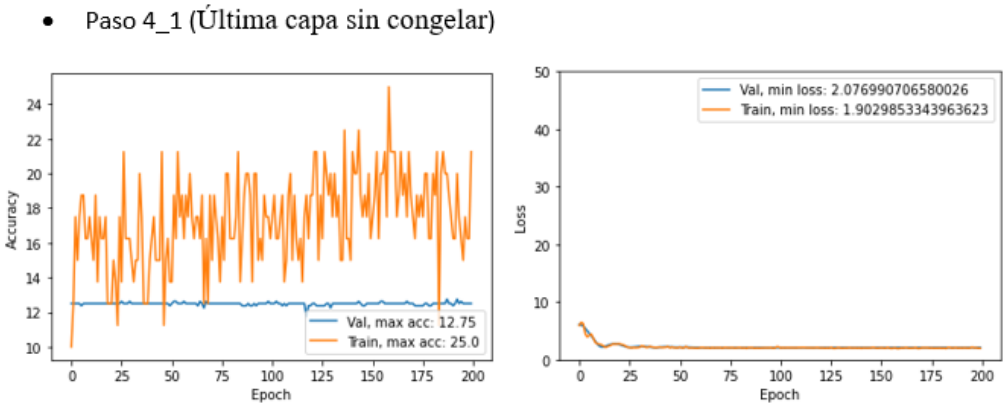



Figura 4.11: Resultado gráfico en 200 épocas representando la precisión y pérdida del Paso 4_1 empleando el conjunto de datos de ISIC 2019.

- Paso 4_2 (75% congelado)

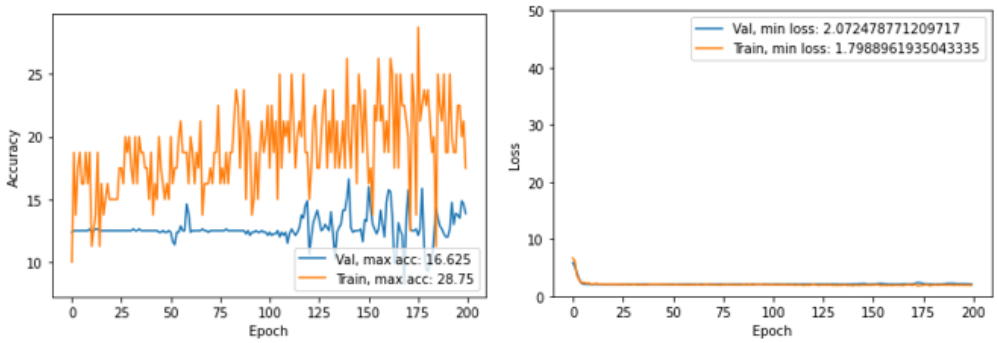



Figura 4.12: Resultado gráfico en 200 épocas representando la precisión y pérdida del Paso 4_2 empleando el conjunto de datos de ISIC 2019.

- Paso 4_3 (50% congelado)

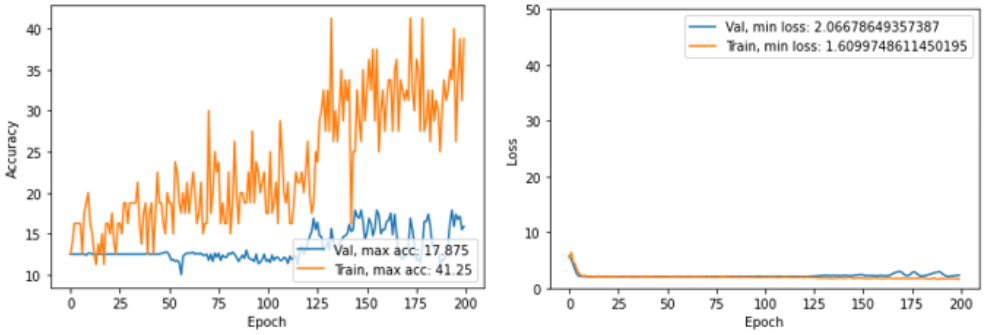



Figura 4.13: Resultado gráfico en 200 épocas representando la precisión y pérdida del Paso 4_3 empleando el conjunto de datos de ISIC 2019.

- Paso 4_4 (25% congelado)

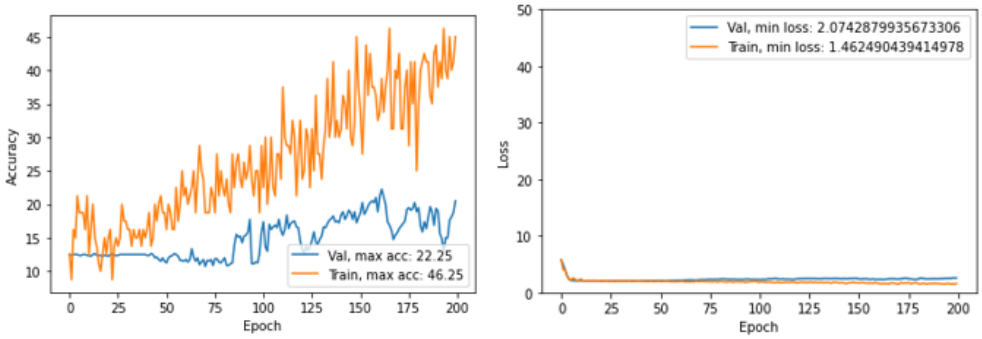



Figura 4.14: Resultado gráfico en 200 épocas representando la precisión y pérdida del Paso 4_4 empleando el conjunto de datos de ISIC 2019.

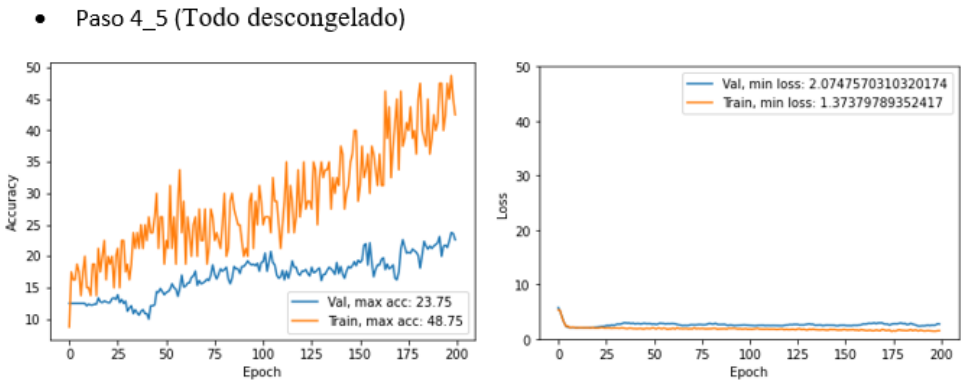



Figura 4.15: Resultado gráfico en 200 épocas representando la precisión y pérdida del Paso 4_5 empleando el conjunto de datos de ISIC 2019.

Por último, se puede comprobar como al descongelar las capas de la red, el modelo presenta mejores resultados. Cuando descongelamos todas las capas, el resultado final es similar al obtenido en el Paso 2. Con esto se puede intuir que los resultados apuntan que la tarea de rotación no ha sido efectiva, ya que el modelo no aprende características generales de la tarea pretexto (no detecta rotaciones).

## 4.4. Análisis, Comparativa y Discusión Final

A continuación, se muestran las pruebas realizadas para ambos conjuntos de datos. Además, las pruebas han sido realizadas de dos formas distintas:

- Con el mismo *learning rate* durante todas las épocas de entrenamiento, el cual es 0.1.
- Con *learning rate* variante durante el entrenamiento. De la época 0 a la 99 se entrena con un *learning rate* de 0.1, de la época 100 a la 149 tiene un valor de 0.01 y por último, de la época 150 a la 199 es de 0.001.

En la primera tabla (Tabla 4.2) podemos observar que los valores cuando varía el *learning rate* son mejores que cuando no lo hace.

Cuando contamos con una gran cantidad de datos la variación del *learning rate* suele tener un efecto positivo.

Por otro lado, se observa que la red cuando está congelada tan solo un 25 % es prácticamente igual que cuando está descongelada por completo. Esto se debe a que en el 25 % de la red solo se han congelado las dos primeras capas, la primera capa convolucional y la primera capa de normalización.

Además, cuando se descongelan las capas se nota una mejora en la precisión de los resultados obtenidos.

**NOTA:** En la **Tabla** 4.2, las tareas son: clasificación imágenes (Cifar, C) y rotación (Pretexto, P).

| Paso | Tarea | Set_T | Set_V | Precisión | Época Máxima Precisión | Precisión con LR variando | Época Máxima Precisión con LR variando |
|---|---|---|---|---|---|---|---|
| Paso1. | C | Set_1_T | Set_1_V | - | - | **91.3 %** | 130 |
| Paso2. | C | Set_2_T | Set_1_V | - | - | **75.31 %** | 190 |
| Paso3. | P | Set_2_T | Set_1_V | **61.19 %** | 60 y 120 | 60.34 % | 141 |
| Paso4. Última capa sin congelar | C | Modelo Paso 3.+ Fine Tuning con Set_2_T | Set_1_V | 48.47 % | 143 | 48.75 % | 136 |
| Paso4. 75 % congelado | C | Modelo Paso 3.+ Fine Tuning con Set_2_T | Set_1_V | 71.1 % | 174 | 72.34 % | 198 |
| Paso4. 50 % congelado | C | Modelo Paso 3.+ Fine Tuning con Set_2_T | Set_1_V | 74.89 % | 196 | **76.61 %** | 189 |
| Paso4. 25 % congelado | C | Modelo Paso 3.+ Fine Tuning con Set_2_T | Set_1_V | 74.65 % | 193 | **77.47 %** | 182 |
| Paso4. Todo descongelado | C | Modelo Paso 3.+ Fine Tuning con Set_2_T | Set_1_V | **75.56 %** | 186 | **77.25 %** | 108 |

Tabla 4.2: Representación de los resultados obtenidos por pasos y de las dos formas distintas en las que se han obtenido para el conjunto de datos CIFAR10.

En la segunda tabla (Tabla 4.3), representamos los resultados obtenidos al igual que para la tabla anterior pero para el conjunto de datos de ISIC 2019.

En esta ocasión, se puede observar una mejora por parte de los resultados cuando no varia el *learning rate*. Esto es debido a que al haber pocos datos en nuestro conjunto, cuando variamos el *learning rate* las curvas tienden a estabilizarse.

**NOTA:** En la **Tabla** 4.3, las tareas son: lesiones de piel (Melanoma, M) y rotación (Pretexto, P).

| Paso | Tarea | Set_T | Set_V | Precisión | Época Máxima Precisión | Precisión con LR variando | Época Máxima Precisión con LR variando |
|---|---|---|---|---|---|---|---|
| Paso1. | M | Set_1_T | Set_1_V | **35 %** | 185 | 30.63 % | 163 |
| Paso2. | M | Set_2_T | Set_1_V | **25.75 %** | 96 | 24 % | 93 |
| Paso3. | P | Set_2_T | Set_1_V | 25.78 % | 55 | **26.469 %** | 196 |
| Paso4. Última capa sin congelar | M | Modelo Paso 3.+ Fine Tuning con Set_2_T | Set_1_V | 12.75 % | 180 | 12.75 % | 192 |
| Paso4. 75 % congelado | M | Modelo Paso 3.+ Fine Tuning con Set_2_T | Set_1_V | 16.625 % | 140 | 15 % | 196 |
| Paso 4. 50 % congelado | M | Modelo Paso 3.+ Fine Tuning con Set_2_T | Set_1_V | 17.875 % | 146 | 15.5 % | 139 |
| Paso 4. 25 % congelado | M | Modelo Paso 3.+ Fine Tuning con Set_2_T | Set_1_V | 22.25 % | 161 | 14.875 % | 187 |
| Paso 4. Todo descongelado | M | Modelo Paso 3.+ Fine Tuning con Set_2_T | Set_1_V | **23.75 %** | 197 | 20.25 % | 134 |

Tabla 4.3: Representación de los resultados obtenidos por pasos y de las dos formas distintas en las que se han obtenido para el conjunto de datos ISIC 2019.

Para finalizar el Capítulo 4, valoramos a través de una discusión final los puntos más importantes.

En primer lugar, el aprendizaje auto-supervisado ha sido de gran utilidad para el desarrollo del trabajo realizado. Gracias a las técnicas auto-supervisadas (tareas pretexto), se puede realizar un aprendizaje con datos no etiquetados, lo que supone una gran ayuda a la hora de trabajar con distintas bases de datos. En nuestro caso, se ha realizado la tarea pretexto de rotación sin ningún tipo de complicación y además, se han visto buenos resultados.

Por otra parte, las técnicas de *transfer learning* y *fine tuning* han sido de gran importancia para la realización de las pruebas. Una vez entrenado el modelo con la tarea pretexto, los resultados sugieren que cuando se descongelan capas, cada vez el efecto es mayor gracias a la *adaptación* de las características aprendidas por la técnica auto-supervisada. Además, gracias a poder congelar y descongelar capas, se pueden adaptar y modelar modelos como se requiera.

Por último, como se ha visto anteriormente, cada arquitectura de red usada ha sido adaptada para cada conjunto de datos de manera distinta, aprovechando las carácterísticas de las imágenes de cada *dataset*. Los resultados parecen indicar que las imágenes dermatoscópicas de lesiones de piel, no son variantes a rotaciones y por tanto, se debe utilizar otra tarea pretexto distinta a la elegida (rotación) en nuestros experimentos.

# 5

# Conclusiones y Trabajo Futuro

## 5.1. Conclusiones

En este Trabajo de Fin de Grado se ha tratado de implementar la técnica de aprendizaje auto supervisado para la clasificación de imágenes. Además, se ha realizado en un dominio no convencional como son las lesiones de piel.

El análisis y estudio del estado del arte ha resultado un punto muy útil, ya que gracias a ello se han obtenido conocimientos acerca de las redes neuronales convolucionales y el papel que juegan. Se ha podido ver las diferentes formas de aprendizaje que hay en las redes neuronales y así poder entender mejor el tipo de aprendizaje utilizado en este trabajo.

Se ha empleado la técnica de *transfer learning* y *fine tuning* con las que hemos podido entender como funciona el proceso de congelación de capas y como el modelo va produciendo mejores resultados conforme se descongelan las capas. El análisis acerca de las bases de datos utilizadas y como la arquitectura de la red utilizada se ha ajustado a ellas ha resultado de gran ayuda para entender el procedimiento que realizan las redes neuronales convolucionales. También, gracias al estudio de los *datasets* se entiende el uso de la tarea auto-supervisada de rotación y su objetivo.

Para el primer conjunto de datos, se puede ver gracias a los resultados gráficos como el uso de la tarea pretexto apunta a una mejora en la clasificación de imágenes. Por otro lado, los resultados del segundo conjunto de datos sugieren el uso de otra técnica auto-supervisada, ya que como se observa las imágenes de lesiones de piel no son variantes a rotaciones. Sin embargo, gracias a la Sección 2.7.1 donde se han propuesto una gran cantidad de técnicas auto-supervisadas se comprende que para diferentes imágenes unas tareas resultarán más o menos efectivas que otras.

Esto nos lleva al siguiente apartado donde el trabajo realizado da lugar a una gran cantidad de trabajos futuros a implementar.

## 5.2. Trabajo Futuro

Como se ha podido observar, los resultados dan lugar a propuestas futuras de mejora para implementarlas en nuestro sistema realizado.

En primer lugar, se podrían probar las diferentes técnicas auto supervisadas para ver cual produce mejores resultados. Así mismo, hacer una comparativa entre ellas para sacar el mayor provecho de cada una de ellas.

También, se podrían implementar diferentes tipos de redes neuronales convolucionales con el fin de extraer características de ellas. Además, una mejora considerable puede ser el aumento drástico de capas en las redes, como puede ser el uso de una ResNet152 que es más compleja y profunda que las utilizadas.

Debido al pequeño número de imágenes utilizadas, sería interesante trabajar con un mayor número de imágenes que incluso aún sufriendo desbalanceo de clases puedan producir mejores resultados. Además, el problema dermatológico de las lesiones de piel es un tema bastante interesante y que puede servir de gran ayuda para prevenir y clasificar rápidamente cualquier problema relacionado.

# Bibliografía


[1] Yu Sun, Eric Tzeng, Trevor Darrell, and Alexei A Efros. Unsupervised domain adaptation through self-supervision. *arXiv preprint arXiv:1909.11825*, 2019. 2

[2] Raúl E. López Briega. Introducción al deep learning. `https://iaarbook.github.io/deeplearning/`, Marzo 2018. Accedido en marzo de 2020. 5, 6, 10

[3] Tarang Shah. Train, validation and test sets. `https://tarangshah.com/blog/2017-12-03/train-validation-and-test-sets/`, Diciembre 2017. Accedido en marzo de 2020. 6

[4] Francis Tseng Andreas Refsgaard and Gene Kogan. Neural networks. `https://ml4a.github.io/ml4a/neural_networks/`, Agosto 2019. Accedido en abril de 2020. 7, 8

[5] Jaime Durán. Todo lo que necesitas saber sobre el descenso del gradiente aplicado a redes neuronales. `https://medium.com/metadatos/todo-lo-que-necesitas-saber-sobre-el-descenso-del-gradiente-aplicado-a-redes-neuronales-19bdbb706a78`, Septiembre 2019. Accedido en abril de 2020. 8, 9, 10

[6] Stanford CS class CS231n: Convolutional Neural Networks for Visual Recognition. Convolutional neural networks (cnns / convnets). *cs231n.github.io/convolutional-networks*, 2019. 10

[7] Diego Calvo. Red neuronal convolucional cnn. `https://www.diegocalvo.es/red-neuronal-convolucional/`, Julio 2017. Accedido en mayo de 2020. 11

[8] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014. 11, 12

[9] Synopsys. Pooling group. `https://embarc.org/embarc_mli/doc/build/html/MLI_kernels/pooling_max.html?fbclid=IwAR3eoZA_08pk`, 2019. Accedido en mayo de 2020. 12

[10] Enrique Dominguez Alfonso Ballesteros. Clasificación de las redes neuronales respecto al aprendizaje. 2018. 13

[11] Yoav Chai, Raja Giryes, and Lior Wolf. Supervised and unsupervised learning of parameterized color enhancement. In *The IEEE Winter Conference on Applications of Computer Vision*, pages 992–1000, 2020. 13

[12] Alexander Kolesnikov, Xiaohua Zhai, and Lucas Beyer. Revisiting self-supervised visual representation learning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 1920–1929, 2019. 13

[13] Lilian Weng. Self-supervised representation learning. *lilianweng.github.io/lil-log*, 2019. 13

[14] Xiaohua Zhai, Avital Oliver, Alexander Kolesnikov, and Lucas Beyer. S4l: Self-supervised semi-supervised learning. In *The IEEE International Conference on Computer Vision (ICCV)*, October 2019. 13

[15] Nikos Komodakis and Spyros Gidaris. Unsupervised representation learning by predicting image rotations. 2018. 14

[16] Alexey Dosovitskiy, Philipp Fischer, Jost Tobias Springenberg, Martin Riedmiller, and Thomas Brox. Discriminative unsupervised feature learning with exemplar convolutional neural networks. *IEEE transactions on pattern analysis and machine intelligence*, 38(9):1734–1747, 2015. 15

[17] Mehdi Noroozi and Paolo Favaro. Unsupervised learning of visual representations by solving jigsaw puzzles. In *European Conference on Computer Vision*, pages 69–84. Springer, 2016. 15

[18] Carl Doersch, Abhinav Gupta, and Alexei A Efros. Unsupervised visual representation learning by context prediction. In *Proceedings of the IEEE international conference on computer vision*, pages 1422–1430, 2015. 15, 16

[19] Mehdi Noroozi, Hamed Pirsiavash, and Paolo Favaro. Representation learning by learning to count. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 5898–5906, 2017. 16

[20] Richard Zhang, Phillip Isola, and Alexei A Efros. Colorful image colorization. In *European conference on computer vision*, pages 649–666. Springer, 2016. 17

[21] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018. 17, 18

[22] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9729–9738, 2020. 18

[23] Grupo de Inteligencia Artificial de Facebook. Documentación de pytorch. 2017. 20

[24] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009. 20, 21

[25] Noel CF Codella, David Gutman, M Emre Celebi, Brian Helba, Michael A Marchetti, Stephen W Dusza, Aadi Kalloo, Konstantinos Liopyris, Nabin Mishra, Harald Kittler, et al. Skin lesion analysis toward melanoma detection: A challenge at the 2017 international symposium on biomedical imaging (isbi), hosted by the international skin imaging collaboration (isic). In *2018 IEEE 15th International Symposium on Biomedical Imaging (ISBI 2018)*, pages 168–172. IEEE, 2018. 21

[26] Philipp Tschandl, Cliff Rosendahl, and Harald Kittler. The ham10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions. *Scientific data*, 5:180161, 2018. 21

[27] Marc Combalia, Noel CF Codella, Veronica Rotemberg, Brian Helba, Veronica Vilaplana, Ofer Reiter, Allan C Halpern, Susana Puig, and Josep Malvehy. Bcn20000: Dermoscopic lesions in the wild. *arXiv preprint arXiv:1908.02288*, 2019. 21

[28] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 23, 24

[29] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. Cnn features off-the-shelf: an astounding baseline for recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 806–813, 2014. 26

[30] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. In *International conference on machine learning*, pages 647–655, 2014. 26

[31] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 3320–3328. Curran Associates, Inc., 2014. 26